

INFORMATICA APLICADA

26

Basic avanzado

AIA

SINCLAIR AMSTRAD COMMODORE IBM



EDICIONES SIGLO CULTURAL

ENCICLOPEDIA PRACTICA DE LA

INFORMATICA APLICADA

26

Basic avanzado

EDICIONES SIGLO CULTURAL

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática
JOSE ARTECHE, Ingeniero de Telecomunicación

Diseño y maquetación:
BRAVO-LOFISH.

Dibujos:
JOSE OCHOA Y ANTONIO PERERA.

Tomo XXVI. **Basic avanzado**
AULA DE INFORMATICA APLICADA (AIA)
JESUS BOCHO, Licenciado en Informática

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta (Ed. Iberia Mart I). Teléf. 810 52 13. 28020 Madrid

Publicidad:
Gofar Publicidad, S.A. San Benito de Castro, 12 bis. 28028 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América. 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-092-8
ISBN de la obra: 84-7688-018-9.

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M. 9.956-1987
Printed in Spain - Impreso en España.

Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta (Ed. Iberia Mart I). Teléf. 810 52 13. 28020 Madrid
Mayo, 1987.
P.V.P. Canarias: 365,-

INDICE



A los usuarios del Spectrum:

En todos los programas de este tomo poner LET.

LA INSTRUCCION PRINT AMPLIADA 1

LOS POSIBLES FORMATOS DE SALIDA CON LA PRINT NORMAL



CUANDO nosotros queremos que el programa nos escriba resultados por pantalla o impresora, normalmente nos interesará que esto lo haga con una disposición determinada; es decir, que nos los escriba en una determinada posición de la pantalla, que nos separe los datos mediante una serie de espacios determinados, que nos escriba únicamente una serie de caracteres de los que forman un dato alfanumérico o que los números que hay guardados en determinadas variables los escriba únicamente con un número determinado de decimales o con notación científica.

DATOS-INFORME

N. ORDEN	%	TASA
1	55.22	9-18
2	37.49	3-49
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.

Fig. 1.

La instrucción PRINT, tal y como la conocemos, nos permite ya realizar algunos de los problemas planteados anteriormente.

Así, ya sabemos que si queremos escribir una serie de datos separados

por un blanco en el caso de que sean numéricos, o sin separación, si son alfanuméricos, basta con que los pongamos en el PRINT separados por un punto y coma. Si los separamos por una coma, nos los escribirá con una separación cuyo número de blancos dependerá de cada equipo con el que se trabaje.

Existen, además, dos funciones y una instrucción que combinadas con la PRINT nos dan la oportunidad de obtener nuevos formatos de salida en pantalla:

— FUNCION SPACE$ (o SPC en otros equipos)

Cuando la ponemos dentro de una instrucción PRINT provoca la impresión de N espacios en blanco. Por ejemplo:

```
PRINT A$;SPACE$(10);B$
```

nos dejaría 10 espacios en blanco entre la palabra que haya en A$ y la que haya en B$.

— FUNCION TAB

Cuando la colocamos dentro de una instrucción PRINT, se salta a la columna cuyo número se ha especificado como argumento de esta función. Así, si ponemos:

```
PRINT TAB(15); N$ ;TAB(15);
```

N$ se escribiría en la columna 15 de la línea actual si es posible. Si esta ya ha pasado, se pone en la columna 15 de la siguiente. El segundo TAB(15) provoca un salto a la posición 15 de la siguiente línea, pues en la actual, la posición 15 ya fue ocupada por N$.

— INSTRUCCION LOCATE

Nos permite situarnos en un lugar concreto de la pantalla para posteriormente, en la siguiente instrucción PRINT, escribir a partir de esa posición donde se ha situado el cursor. Conviene recordar las variantes que presenta esta instrucción en los distintos equipos y que ya fueron expuestas en el anterior número.

La siguiente figura nos muestra un resumen de las distintas posibilidades de formatos de salida utilizando la instrucción PRINT.

PRINT A;B	**Separa los n.os de A y B por un espacio**
PRINT A$;B$	**No separa los valores de A$ y B$**
PRINT A$,B	**Separa los valores con un n.º de espacios**
LOCATE (PRINT AT en SPECTRUM)	**Se sitúa en unas coordenadas de la pantalla**
SPACE$(N)(SPC(N))	**Escribe N espacios en blanco**
TAB(N)	**Se sitúa en la próxima columna N a partir de donde está el cursor**

Fig. 2.



TENEMOS MAS POSIBILIDADES CON LA INSTRUCCION PRINT USING

Con lo que hemos visto hasta ahora, hay algunos posibles formatos de salida que nos gustaría obtener, y que no sabemos cómo hacerlo, sobre todo en lo que respecta a la escritura de números. Cuando en una instrucción PRINT, mandamos escribir el número que hay guardado en una variable de su mismo tipo, nos lo escribe siempre con las siguientes condiciones:

a) Los números negativos llevan siempre delante el signo.
b) En los positivos el espacio del signo queda en blanco.
c) Detrás de cada número aparece siempre un espacio en blanco.

Existe, sin embargo, una modalidad de la instrucción PRINT que permite controlar con precisión cómo se van a imprimir los números en cuanto a tratamiento de ceros, puntos y comas decimales. Se trata de la instrucción PRINT USING, cuyo formato es el siguiente:

PRINT USING dato alfanumérico; variables o datos que se quieren escribir.

Se diferencia de la instrucción PRINT normal, en que además de añadirle la palabra USING, antes de poner los datos que se quieren escribir, se debe colocar un dato alfanumérico (bien como una constante entre comillas o mediante el nombre de la variable donde se encuentra guardado dicho dato). Este dato estará formado por una serie de caracteres especia-

les que van a tener un efecto en la escritura de lo que venga a continuación y cuyo significado se explica seguidamente:

a) *Cuando los datos que vamos a escribir en el PRINT son alfanuméricos:*

— ! Este carácter indica que solo se imprima la primera letra del dato alfanumérico que venga a continuación.

— \\ Se imprimen dos caracteres del dato más uno por cada espacio en blanco que se ponga entre las barras.

— & Se imprime el dato alfanumérico íntegramente.

Supongamos, pues, que tenemos en las variables N$ y M$ el caso de la figura y se ejecutan las siguientes instrucciones:

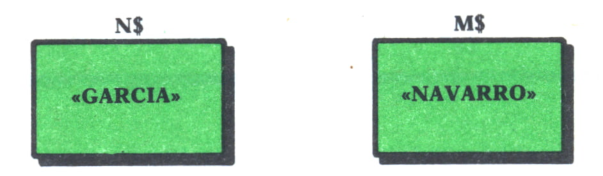


Fig. 3.

```
10 REM EJEMPLO DE FORMATOS PARA DATOS ALFANUMERICOS
20 REM ******************************************
30 N$="GARCIA":M$="NAVARRO"
40 PRINT USING "!";N$;M$
50 PRINT USING "\    \ ";N$;M$
60 PRINT USING "& ";N$;M$
```

Programa 1.

Al ejecutarse el programa, el primer PRINT USING escribirá las iniciales de ambas palabras, el segundo escribirá las seis primeras letras de éstas, ya que hemos colocado cuatro espacios en blanco entre las barras; y el tercero nos sacará por pantalla las palabras íntegramente. Es interesante notar que en los dos últimos se ha colocado un espacio en blanco al final de los caracteres alfanuméricos que especifican el formato. Esto se ha hecho para que al escribir las palabras que vengan a continuación, se haga dejando un espacio en blanco entre ellas. Si se quisieran dejar más espacios en blanco de separación entre las palabras, bastaría con dejar más espacios en blanco al final.

b) *Cuando queremos escribir datos numéricos*

— # Se pone un carácter de éstos por cada cifra que queramos que nos escriba de los números que vengan a continuación. Si el número a vi-

sualizar tiene menos dígitos que cifras especificadas, se alineará a la derecha (precedido de espacios en blanco).

También podemos colocar el punto decimal donde queramos que nos lo escriba.

Veamos algunos ejemplos de estas posibilidades:

Vamos a suponer para todos ellos que tenemos, en el momento de ser ejecutadas las instrucciones, en las variables los valores especificados en la figura:

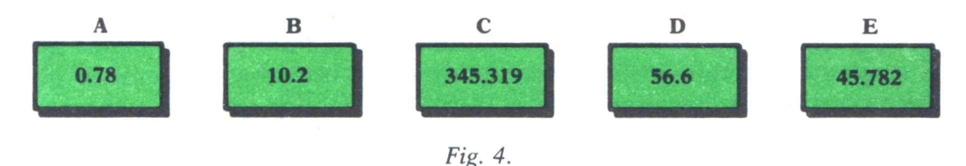


Fig. 4.

PRINT USING "##.## ";A;B

nos escribiría:

0.78 10.20

PRINT USING "###.## ";C;D;E

nos escribiría:

345.32   56.60   45.78

En el último ejemplo se han colocado dos espacios al final del dato alfanumérico que nos describe el formato, para separar los valores por este número de espacios. Además, en los números que únicamente tienen dos cifras enteras nos deja un blanco en el lugar que correspondería a la cifra de las centenas que no llevan. Si quisiéramos dejar una serie de espacios en blanco delante del número, bastaría con que los pusiésemos delante de los caracteres Pt del formato. En aquellos datos numéricos que tienen más de dos cifras decimales se redondean éstas.

— Signos + – :

Un signo más delante o detrás del dato alfanumérico de formato provoca que el signo del número que se escriba se visualice delante o detrás de dicho número. Un signo – al final provoca que se visualice el signo únicamente cuando éste es negativo y se haga al final de él.

— ^^^^

Cuatro caracteres de este tipo al final del dato de formato indican que los números deben escribirse en notación científica; formato exponencial.

Por ejemplo:

Si ponemos la instrucción:

PRINT USING “##.##^^^^“;D

nos escribiría:

5.66E+1 (suponiendo que en D tenemos guardado el mismo dato del ejemplo anterior).

— Signo

Una raya debajo dentro del valor alfanumérico de formato indica que el siguiente carácter que viene a continuación de ésta es un carácter que debe visualizarse literalmente tal y como está. Si, por ejemplo, ponemos:

PRINT USING“_!##.##_!“;D

nos escribiría:

!56.60!

Como aplicación de este último caso visto, ofrecemos un programa que nos escribiría los números decimales con coma, tal y como se hace normalmente en España en vez del punto decimal.

```
10 REM ESCRITURA DE DECIMALES CON COMA
20 REM ******************************
30 INPUT "Introduce un número que contenga decimales";N
40 PRINT USING "#####_´"; INT(N) ;
50 PRINT USING "#####"  ;(N-INT(N))*100000!
60 REM El programa escribe números hasta con 5 cifras decimales
70 REM Para escribir números con mas cifras, habrá que aumentar
80 REM el número de # en 50 y el número de ceros de la cifra por
90 REM la que se multiplica al final de 50
100 REM Para disminuir las cifras, lo contrario
```

Programa 2.

Existen otros caracteres especiales de formato que permiten añadir asteriscos u otros signos. No vamos a estudiarlos aquí por considerarlos de un uso muy restringido. De todas formas, la técnica es la misma y esperamos que con las ideas aquí ofrecidas, si algún lector desea usar algunos de ellos, lo pueda hacer consultando el manual del equipo correspondiente. De igual forma, se aconseja consultar el manual si en alguno de los caracteres aquí explicados algún equipo presentase alguna variante, aunque éstos son estándar para la gran mayoría de los ordenadores.

El siguiente cuadro es un resumen de los caracteres especiales de formato estudiados:

CARACTER	FUNCION
!	**Escribe la inicial de los datos alfanuméricos**
\\	**Imprime dos caracteres, más uno por cada blanco entre medias, de los datos alfanuméricos**
&	**Imprime los datos alfanuméricos íntegramente**
#	**Imprime una cifra de los números por cada carácter de éstos**
+	**Imprime el signo del número en el lugar que se coloque**
-	**Imprime sólo el signo negativo en el lugar que se coloque**
^^^^	**Imprime el n.º en notación científica**
_	**Imprime el siguiente carácter tal y como viene en la posición colocada entre las cifras del n.º**

Fig. 5.

A continuación se ofrece un programa aplicación de los formatos de salida estudiados en este capítulo:

```
10 REM PORCENTAJE DE VOTOS DE UNA SERIE DE PARTIDOS TRAS UNA VOTACION
20 REM ***********************************************************
30 CLS
40 INPUT "introduce número de partidos";N
50 DIM N$(N,5) :DIM V(N)
60 INPUT "INTRODUCE EL CENSO TOTAL VOTANTE" ;CEN
70 FOR I=1 TO N
80 PRINT "TECLEE A CONTINUACION EL NOMBRE DEL PARTIDO N.";I
90 PRINT "SI ESTA COMPUESTO POR VARIAS PALABRAS, PONGA UNA EN CADA LINEA"
100 FOR J=1 TO 5
110 PRINT
120 INPUT "PALABRA  DEL NOMBRE ( PULSE RETURN SI NO HAY MAS)";N$(I,J)
130 NEXT J
140 INPUT "VOTOS=",V(I)
150 CLS
160 NEXT I
170 REM IMPRESION DE LOS PORCENTAJES
180 REM ****************************
190 PRINT "PORCENTAJE DE VOTOS DE CADA PARTIDO":PRINT
200 FOR I=1 TO N
```

```
210 FOR J=1 TO 5
220 PRINT USING "! ";N$(I,J);
230 NEXT J
240 PRINT USING "  ##.##_%"; V(I)*100/CEN ;
250 PRINT "  DE LOS VOTOS"
260 NEXT I
```

Programa 3.

EJERCICIOS:

1. ¿Qué diferencia hay entre la función SPACE$ (o SPC) y la función TAB?

2. Supongamos que A=3.14159 y B=–6.8976598. ¿Cómo pondríamos una instrucción PRINT USING para que nos los escriba en una misma fila, a partir de la columna 3, con seis espacios en blanco de separación entre ellos, dos decimales y su signo correspondiente?

3. Supongamos que en una variable N$ tenemos el nombre de pila de una persona que hemos obtenido dentro de un programa de una lista en la que tenemos los nombres de una serie de personas. Realizar un trozo de programa para escribir este nombre con la condición de que en aquellos nombres que sean compuestos, únicamente escriba el primer nombre completo y la inicial del segundo. Un nombre compuesto vendrá determinado porque entre el primer nombre que lo componga y el segundo habrá un espacio en blanco.

RESPUESTAS:

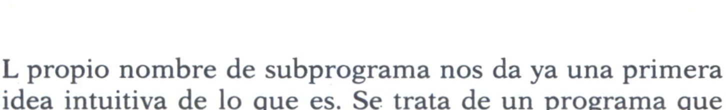
2. PRINT USING“ +#.## “;A;B

3.

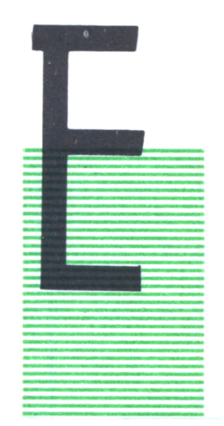
```
10 REM Antes de ejecutar este trozo, en N$ debe de haber un nombre
20 I=1
30 A$=MID$(N$,I,1)
40 IF A$=" " THEN      GOTO 100
50 REM Si encontramos un blanco es que hay nombre compuesto
60 IF A$=""  THEN PRINT N$ :END
70 REM Si encontramos el final sin haber encontrado blanco es que no hay nombre
compuesto
80 I=I+1
90 GOTO 30
100 F$="\"
110 REM I+1 es el número de caracteres de N$ que hay que escribir
120 REM habrá que poner por tanto I-1 blancos entre las barras
130 FOR J=1 TO I-1
140 F$=F$+" "
150 NEXT J
160 F$=F$+"\"
170 PRINT USING F$ ; N$
```

Programa 4.

LAS SUBRUTINAS O SUBPROGRAMAS 2



¿QUE ES UN SUBPROGRAMA?



EL propio nombre de subprograma nos da ya una primera idea intuitiva de lo que es. Se trata de un programa que se va a encontrar incluido dentro de otro programa mayor.

Veamos, sin embargo, cuando nos surge la necesidad de utilizar subprogramas y cómo se utilizan éstos. A menudo, al hacer un programa, se nos puede plantear el siguiente problema: hay una serie de instrucciones consecutivas que queremos realizar exactamente igual en distintas partes del programa.

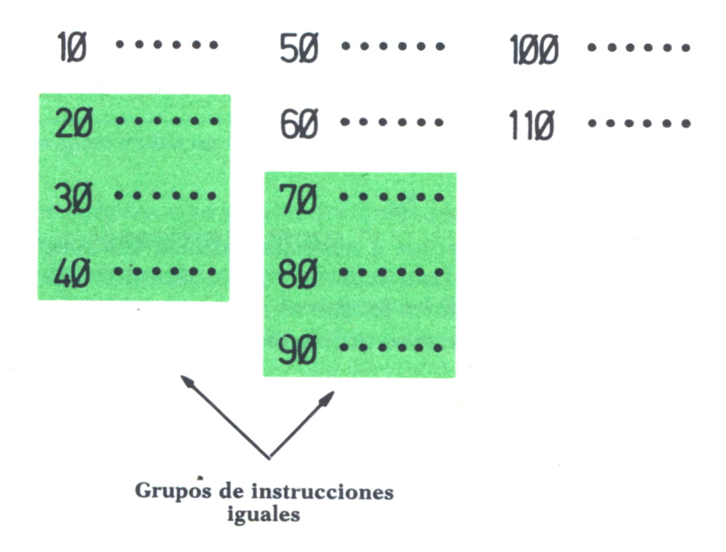


Fig. 1.

Con las instrucciones conocidas hasta ahora, no tenemos otra solución que escribir el grupo de instrucciones iguales en todos los momentos del programa que se vayan a realizar, ya que al no repetirse una detrás de otra, sino en diferentes momentos, no existe ninguna manera posible de agruparlas dentro de una instrucción FOR.

Tampoco sería factible la solución de colocar estas instrucciones en un lugar determinado del programa y saltar a ellas con un GOTO cada vez que queramos que el programa las ejecute, ya que después que se terminen de ejecutar querríamos que el programa volviese a un lugar distinto cada vez, según desde el sitio en que se saltara a ejecutarlas.

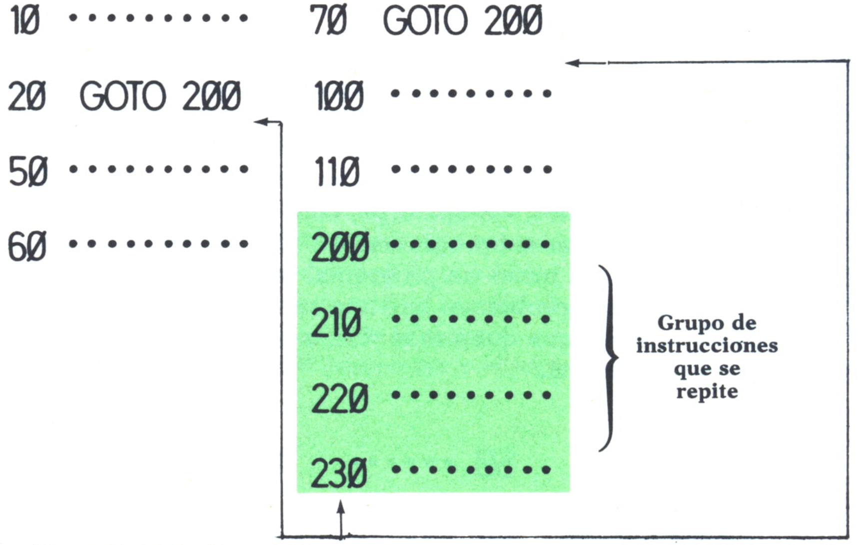


Fig. 2.

Este es el caso en el cual vamos a necesitar utilizar los subprogramas o subrutinas. Este conjunto de instrucciones que queremos repetir en varios sitios distintos del programa es lo que se llama un SUBPROGRAMA y a continuación vamos a ver cómo se utiliza en un programa BASIC.



LA INSTRUCCION GOSUB

Cuando en un programa se nos plantea el caso de utilizar uno o más subprogramas, lo primero que se debe hacer es escribir estos al final del programa a partir de un número al que no puedan llegar nunca las instrucciones de nuestro programa. El subprograma también podría escribir-

se en cualquier parte del programa, pero esto exigiría un mayor cuidado y control de lo que haría el programa al ser ejecutado, y al mismo tiempo daría lugar a un programa mucho más confuso a la hora de tratar de ser entendido por alguien que no lo haya hecho. Por esta razón, se aconseja que los subprogramas en BASIC se escriban siempre al final del programa y en todos los ejemplos que se presenten en este libro así se hará.

Al final del subprograma se debe poner la instrucción RETURN, que indica precisamente que se ha acabado esa subrutina.

Una vez que hemos escrito la subrutina, cada vez que en el programa queramos, en un lugar determinado, ejecutar las instrucciones del subprograma tendremos que poner la instrucción:

GOSUB número de línea

El número de línea será aquella en la que comience la subrutina y cuando el programa llegue a una instrucción de este tipo, ejecutará las instrucciones de la subrutina a partir del número de línea especificada hasta que encuentre la instrucción RETURN, que indica que se ha acabado la subrutina; VOLVIENDO entonces a EJECUTAR LA INSTRUCCION SIGUIENTE a la GOSUB desde la que saltó al subprograma.

```
10...              70
20...              80 END
30 GOSUB 1000      1000
40...              1010
50...              1020
60 GOSUB 1000      1030
                   1040
                   1050 RETURN
```

Fig. 3.

La figura anterior es solamente un esqueleto de programa para ilustrar la mecánica del GOSUB-RETURN.

Tenemos un programa principal que comprende desde la línea 10 a la 80 y una subrutina en las instrucciones 1000-1050.

El programa ejecutaría las instrucciones en el siguiente orden: 10-20-30-1000-1010-1020-1030-1040-1050-40-50-60-1000-1010-1020-1030-1040-1050-70-80.

En las instrucciones 30 y 60 encuentra un GOSUB 1000, por lo que salta a ejecutar la subrutina que comienza en 1000 y cuando en 1050 encuentra la instrucción RETURN, vuelve a la siguiente instrucción de la GOSUB (40 y 70, respectivamente).

A continuación se ofrece un programa en el que se puede observar el empleo de subrutinas:

```
10 REM RESERVA DE PLAZAS EN UN TREN
20 REM ***************************
25 REM La lista P tendrá un 0 si la plaza está libre
26 REM y un 1 si está ocupada.Hay tres vagones cada
27 REM uno con 10 plazas
30 DIM P(30)
40 GOSUB 1000
50 INPUT "¿EN QUE VAGON SE QUIERE RESERVAR";V
55 IF V=0 THEN GOTO 110
60 K=(V-1)*10+1
65 REM K indica la primera plaza del vagón solicitado
70 FOR I=K TO K+9
80 IF P(I)=0 THEN P(I)=1 :PRINT "SE RESERVA LA PLAZA ";I:GOTO 40
90 NEXT I
100 GOTO 40
110 GOSUB 1000
120 END
900 REM LA SUBRUTINA LISTA POR PANTALLA LAS PLAZAS OCUPADAS
1000 CLS
1005 PRINT "PLAZAS OCUPADAS"
1010 PRINT "VAGON 1"
1020 FOR J=1 TO 10
1030 IF P(J)=1 THEN PRINT "PLAZA ";J
1040 NEXT J
1045 PRINT
1050 PRINT "VAGON 2"
1060 FOR J=11 TO 20
1070 IF P(J)=1 THEN PRINT "PLAZA ";J
1080 NEXT J
1085 PRINT
1090 PRINT "VAGON 3"
1100 FOR J=21 TO 30
1110 IF P(J)=1 THEN PRINT "PLAZA ";J
1120 NEXT J
1130 RETURN
```

Programa 1.

En los programas en los que se emplean subrutinas, hay que tener en cuenta un pequeño detalle. Cuando nosotros escribimos un programa normal, no es necesario que pongamos al final de éste la instrucción que indica que el programa se ha acabado (END), ya que si el ordenador no encuentra más instrucciones que ejecutar, acaba el programa de todas maneras. Sólo tenemos que poner la instrucción END, en el caso de que el programa se acabase a la mitad cuando se cumpla una determinada condición. En un programa en el que usemos subrutinas, sí tenemos que poner como última instrucción del programa la instrucción END, ya que si no el programa continuaría ejecutando la subrutina, que está puesta a continuación, realizando cosas erróneas.

Los subprogramas permiten la modularización de los programas. Podemos dividir un programa largo, en uno principal y una serie de subrutinas para realizar diversas operaciones. Alguna de éstas puede ser luego copiada a otros programas que realicemos. Cuando hagamos un programa grande es aconsejable realizarlo siempre a través de subrutinas, es decir, hacer una subrutina por cada operación que tenga que realizar el programa, aunque se haga una única vez en éste. Es una manera de conseguir que el programa sea lo más inteligible posible y de acercarnos en BASIC a la idea de programación ESTRUCTURADA (programar por módulos que luego se ensamblan). En los programas de ejemplo que se ponen más adelante veremos algún caso de esto.



LOS ARGUMENTOS DE LAS SUBRUTINAS

Vamos a ver en este apartado una idea fundamental en el empleo de subrutinas que en otros lenguajes de programación viene implícita en la propia definición de subrutina y que en BASIC tenemos que realizarla nosotros con una serie de instrucciones.

Veamos la idea con un ejemplo muy sencillo:

Supongamos que queremos hacer un programa para calcular el área de unas figuras con la misma forma que la del dibujo y comparar sus secciones con un par de valores:

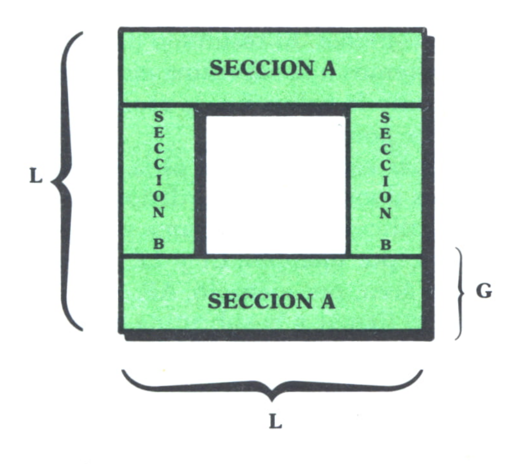


Fig. 4.

A partir de los datos L y G el programa deberá obtener la superficie total de la figura y las superficies de las secciones A y B. También queremos que nos diga por pantalla si los valores de estas secciones son mayores que

el dato que tenemos guardado en la variable MAX y si son menores que el que hay guardado en la variable MIN.

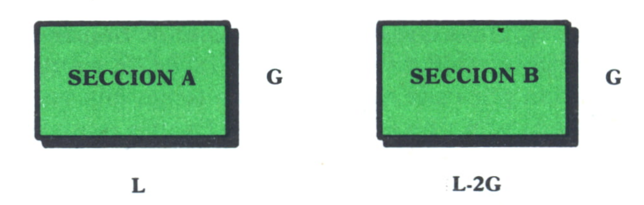


Fig. 5.

El programa consistirá, pues, en hallar el área de dos rectángulos (A y B) y compararla con dos valores. Luego el área total de la figura será el doble de la suma de A y B.

Supongamos que pensamos definir una subrutina para calcular el área de un rectángulo y compararla con los dos valores mencionados. Luego el programa llamará dos veces a la subrutina para calcular el área de las dos secciones y compararla con los valores máximo y mínimo.

El problema nos surge al escribir la subrutina. Para calcular el área, por ejemplo, tendremos que multiplicar las variables donde se encuentren guardados los valores de los lados.

¿Qué variables multiplicamos en la subrutina? L y G, que son las que tenemos que multiplicar cuando saltemos a la subrutina para calcular el área del primer rectángulo o (L-2*G)* G que sería la multiplicación que tendríamos que hacer para calcular el área del segundo rectángulo.

Aún nos surge otro problema. El resultado del área queremos guardarlo la primera vez que saltemos a la subrutina en la variable A y la segunda vez en la variable B y como la subrutina se escribe una sola vez evidentemente, a ver qué variable ponemos.

La forma de solucionar este problema es la siguiente: en la subrutina ponemos como variables donde están guardados los lados del rectángulo, dos variables que no se utilicen en el programa y que van a ser propias únicamente de la subrutina; por ejemplo, L1 y L2. Lo mismo hacemos para guardar el resultado del área; por ejemplo, AR.

Una vez hecho esto, al llamar a la subrutina para calcular el área de la sección A deberemos poner:

```
70 L1=L :L2=G
90 GOSUB 1000
100 A=AR
```

Programa 2.

Haciendo esto, conseguimos que antes de saltar a la subrutina las variables L1 y L2 tomen los valores de los lados del rectángulo cuya área queremos obtener y cuando termine la subrutina (que tiene el resultado en AR) se pasa este resultado a la variable A, que es donde queremos guardarlo. Para calcular el área de la sección B, se haría de manera similar:

```
110 L1=L-(2*G) :L2=G
130 GOSUB 1000
140 B=AR
```

Programa 3.

Hay que notar también que en este programa tendríamos que hacer lo mismo con otra variable. Si nosotros queremos que cuando se cumplan las condiciones de que las secciones sean mayores o menores que los límites, nos escriba un mensaje del estilo:

LA SECCION A ES MAYOR QUE EL MAXIMO PERMITIDO

Como este mensaje lo escribiremos en la subrutina, tampoco podríamos poner ni **SECCION A** ni **SECCION B**, ya que unas veces lo haremos para la **A** y otras veces para la **B**. Esto se soluciona también poniendo una variable en la que guardamos la letra según saltemos a la subrutina para calcular un área u otra.

Veamos a continuación este programa completo:

```
10 REM AREA DE LA FIGURA
20 REM *****************
25 CLS
30 INPUT "VALOR DE L=",L
40 INPUT "VALOR DE G=", G
50 INPUT "VALOR MAXIMO PERMITIDO DE LA SECCION"; MAX
60 INPUT "VALOR MINIMO PERMITIDO DE LA SECCION";MIN
70 L1=L :L2=G
80 A$="A"
90 GOSUB 1000
100 A=AR
110 L1=L-(2*G) :L2=G
120 A$="B"
130 GOSUB 1000
140 B=AR
150 PRINT "EL AREA TOTAL ES "; 2*(A+B)
160 END
1000 AR=L1*L2
1010 PRINT "EL AREA DE LA SECCION ";A$;" ES";AR
1020 IF AR>MAX THEN PRINT "LA SECCION "; A$;" ES MAYOR QUE EL MAXIMO PERMITIDO"
1030 IF AR<MIN THEN PRINT "LA SECCION "; A$;" ES MENOR QUE EL MINIMO PERMITIDO"
1040 RETURN
```

Programa 4.

Estas variables que utilizamos de esta forma son lo que se suelen llamar argumentos del subprograma.

Otro programa que también utiliza subprogramas con la misma idea es el que transforma un número romano en un número decimal, que se ofrece a continuación.

La subrutina se encarga de transformar una letra en su código numérico correspondiente. Se utiliza dos veces, ya que cada vez que analizamos una letra hay que obtener también el valor de la siguiente para saber si hay que sumar o restar, según sea mayor o menor. Las variables A$ y A son los argumentos de la subrutina.

```
10 CLS
20 INPUT "DAME EL NUMERO ROMANO ";R$
30 FOR X=1 TO LEN(R$)
40 L$=MID$(R$,X,1)
50 A$=L$:GOSUB 300
60 L=A
70 S$=MID$(R$,X+1,1)
80 A$=S$:GOSUB 300
90 N=A
100 IF L>=N THEN Z=Z+L ELSE Z=Z-L
110 NEXT X
120 LOCATE 12,5:PRINT "EL RESULTADO ES ";Z
140 END
300 IF A$="M" THEN A=1000
310 IF A$="D" THEN A=500
320 IF A$="C" THEN A=100
330 IF A$="L" THEN A=50
340 IF A$="X" THEN A=10
350 IF A$="V" THEN A=5
360 IF A$="I" THEN A=1
365 IF A$="" THEN  A=0
370 RETURN
```

Programa 5.



LA INSTRUCCION ON..GOSUB

Cuando hablamos de la decisión múltiple en el número anterior, presentábamos la instrucción ON .. GOTO.

La instrucción ON .. GOSUB que vemos seguidamente amplía las posibilidades de la ON .. GOTO, ya que nos evita el tener que poner, al final de cada instrucción o grupo de instrucciones a las que saltamos mediante el ON .. GOTO, una instrucción GOTO para ir a un punto común del programa.

```
10 ON N GOTO 20, 30, 40
20... GOTO 50
30... GOTO 50
40... GOTO 50
50...
```

```
10 ON N GOSUB 20, 30, 40
15...
20...
30...
40...
```

En la ON.. GOTO necesitamos indicar el salto a donde queremos que continúe el programa

Fig. 6.

La instrucción ON .. GOSUB tiene el siguiente formato:

ON variable GOSUB L1,L2,L3,....

El funcionamiento es exactamente el mismo que el de la instrucción ON .. GOTO, con la diferencia de que las líneas L1,L2,L3,..a las que salta según los valores de la variable son líneas de comienzo de subrutinas. Cuando desde esta instrucción se salta a una subrutina, se ejecutan las instrucciones en el orden en que estén hasta que llegue a la instrucción RETURN y VOLVIENDO ENTONCES A EJECUTAR LA INSTRUCCION QUE HAYA A CONTINUACION DEL ON .. GOSUB.



Fig. 7.

El siguiente programa es un ejemplo de utilización del ON .. GOSUB:

```
10 REM control de stock de un almacen
20 REM ******************************
30 DIM A(10)
35 CLS
40 PRINT "MENU" :PRINT
50 PRINT "INICIALIZAR STOCK DE CADA PRODUCTO        -->1"
60 PRINT "CAMBIAR STOCK DE UN PRODUCTO              -->2"
70 PRINT "VISUALIZAR STOCK DE TODOS LOS PRODUCTOS -->3"
75 PRINT "TERMINAR                                  -->4"
80 PRINT :INPUT "TECLEA OPCION"; N
85 CLS
90 ON N GOSUB 500,700,900
100 IF N<>4 THEN GOTO 40
110 END
500 FOR I=1 TO 10
510 PRINT "STOCK DEL PRODUCTO"; I
```

```
520 INPUT A(I)
530 NEXT I
540 RETURN
700 INPUT "N. DE PRODUCTO DEL QUE HA HABIDO CAMBIO";P
710 INPUT "CAMBIO DE STOCK CON SU SIGNO";C
720 A(P)=A(P)+C
730 RETURN
900 FOR I=1 TO 10
910 PRINT "STOCK DEL PRODUCTO";I;"="; A(I)
920 NEXT I
930 RETURN
```

Programa 6.

PROGRAMAS APLICACION DEL GOSUB

```
10 REM CALCULO DE LAS COMBINACIONES DE N ELEMENTOS TOMADOS DE M EN M
20 REM ************************************************************
30 CLS
40 INPUT "INTRODUCE LOS VALORES DE N Y M";N,M
50 X=N
60 GOSUB 1000
70 FACN=F
80 X=M
90 GOSUB 1000
100 FACM=F
101 X=N-M
102 GOSUB 1000
103 FACNM=F
110 PRINT "EL RESULTADO ES";FACN/(FACM*FACNM)
120 END
1000 REM SUBRUTINA QUE HALLA EL FACTORIAL DEL NUMERO GUARDADO EN X
1010 REM EL FACTORIAL LO GUARDA EN F
1015 F=1
1020 FOR I=2 TO X
1030 F=F*I
1040 NEXT I
1050 RETURN
1060 REM Se deja como ejercicio al usuario el control de que no se
1070 REM introduzcan números cuyo factorial desborde la capacidad
1080 REM de cálculo del ordenador
```

Programa 7.

```
10 REM PASO DE UNA FECHA NUMERICA A TEXTO
20 REM ********************************
25 CLS
30 DIM M$(12)
40 GOSUB 1000
50 INPUT "FECHA (D,M,A)";D,M,A
60 GOSUB 2000
```

```
62 REM E=1 INDICA QUE LA FECHA ES ERRONEA
65 IF E=1 THEN E=0:GOTO 50
70 PRINT "DIA:";D
80 PRINT "MES:";M$(M)
90 PRINT "AÑO:";A
100 END
1000 DATA "ENERO","FEBRERO","MARZO","ABRIL","MAYO","JUNIO","JULIO"
1010 DATA "AGOSTO","SEPTIEMBRE","OCTUBRE","NOVIEMBRE", "DICIEMBRE"
1020 FOR I=1 TO 12
1030 READ M$(I)
1040 NEXT I
1050 RETURN
2000 IF D>31 THEN PRINT "FECHA ERRONEA":E=1:RETURN
2010 IF D>28 AND M=2 THEN PRINT "FECHA ERRONEA":E=1:RETURN
2020 IF D=31 AND (M=4 OR M=6 OR M=9 OR M=11) THEN PRINT "FECHA ERRONEA":E=1:
RETURN
2030 IF M>12 THEN PRINT "FECHA ERRONEA":E=1:RETURN
2040 RETURN
```

Programa 8.

En este último programa es importante destacar la utilización de las subrutinas para realizar dos módulos del programa: almacenamiento en las variables de los datos incluidos en el DATA y validación de la fecha introducida. Esto estaría dentro de la idea de utilizar las subrutinas para realizar con cada una un módulo de un programa y luego saltar a estos módulos cada vez que queramos realizar las operaciones que les correspondan. Es una forma de acercarnos con el BASIC a las ideas de la programación estructurada, como ya dijimos anteriormente.

EJERCICIOS:

1. ¿Cómo se sabe dónde empieza y dónde termina un subprograma BASIC?

2. Realizar el programa para escribir con letras el valor del número que hay guardado en una variable (que aparecía en el número anterior) utilizando instrucciones ON .. GOSUB

3. Intenta averiguar, antes de escribir RUN, lo que hará el siguiente programa si se introducen por pantalla los números 5 y 2:

```
10 INPUT "Introduce un número";A
20 INPUT "Introduce otro número";B
30 C=20
40 D=10
50 X=A:Y=B
60 GOSUB 140
70 X=C:Y=D
80 GOSUB 140
```

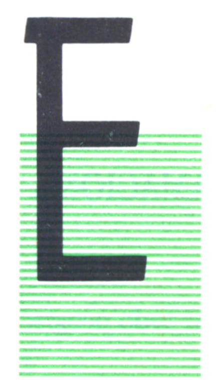
```
90 X=A:Y=C
100 GOSUB 140
110 END
140 SUMA=X+Y
150 MULT=X*Y
160 PRINT SUMA,MULT
170 RETURN
```

Programa 9.

3 LA PROGRAMACION ESTRUCTURADA



INTRODUCCION



EN 1968 Dijkstra publicó un artículo titulado: «*Go to statement considered harmful*», que se puede traducir de una manera más o menos libre por: «La instrucción GOTO es peligrosa». En él afirma que la calidad de un programador es inversamente proporcional al número de instrucciones GOTO que usa en sus programas y que dicha instrucción debería suprimirse de los lenguajes de programación de alto nivel.

La controversia planteada por la publicación de este artículo hizo que se prestase una mayor atención a la organización interna de los programas, con la idea de hacerlos más simples y libres de errores.

La idea fundamental se basa en conseguir realizar un programa a base de ensamblar una serie de estructuras independientes. La instrucción GOTO, por tanto, rompe esta estructura, ya que provoca saltos de unos lugares a otros y hace que no se puedan realizar estructuras independientes una tras otra.

```
10...
20...
30... GOTO 10
40...
50... GOTO 20
60... GOTO 80
70...
80... GOTO 10
```

El GOTO hace que no se pueda dividir el programa en módulos independientes

Fig. 1.

A partir de esto, surge la idea de la programación ESTRUCTURADA. Los métodos de la programación estructurada lo que hacen es definir una serie de estructuras tipo a partir de las cuales y únicamente con ellas se

puede escribir cualquier programa a base de ir ensamblando una tras otra estructuras de este tipo (en el orden que queramos).

En este capítulo vamos a ver cuáles son estas estructuras y cómo se podrían realizar en BASIC. De todas formas, veremos que sólo se pueden realizar en BASIC en algunos equipos que cuenten con algunas instrucciones especiales, ya que se trata de un lenguaje que no está orientado a la programación estructurada. Esto será fácil de notar por el lector, ya que es de suponer que se estará imaginando que es imposible hacer determinados programas en BASIC sin utilizar la instrucción GOTO. Veremos que sólo será posible en aquellos ordenadores que cuenten con la instrucción WHILE-WEND.

LAS ESTRUCTURAS TIPO DE LA PROGRAMACION ESTRUCTURADA

Las estructuras tipo a partir de las cuales se debe realizar todo programa estructurado son las siguientes:

a) *Estructura de bloque*

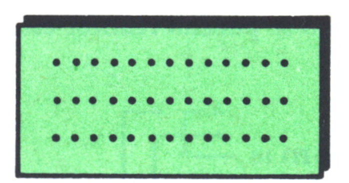


Fig. 2.

Consiste en un conjunto de instrucciones o estructuras de las que se estudien a continuación ordenadas que siempre se ejecutan en el mismo orden y que en conjunto realizan una determinada operación que nos resulta interesante para lo que quiere hacer nuestro programa.

b) *Estructura IF-THEN-ELSE*

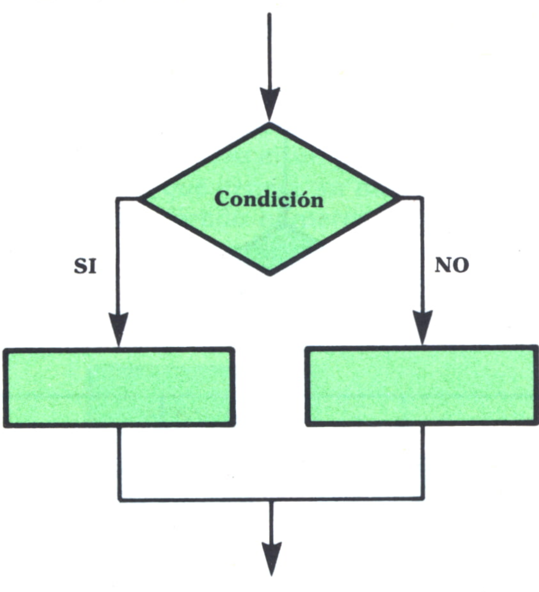


Fig. 3.

Consiste en realizar una instrucción o una estructura completa en el caso de que se cumpla una determinada condición, y otra instrucción u otra estructura si no se cumple esta condición.

c) *Estructura DO-WHILE*

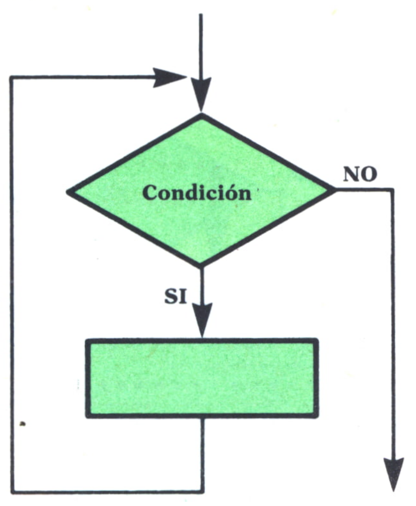


Fig. 4.

Consiste en repetir una serie de instrucciones o estructuras ordenadas, siempre en el mismo orden, mientras se cumpla una determinada condición.

d) *Estructura caso (case)*

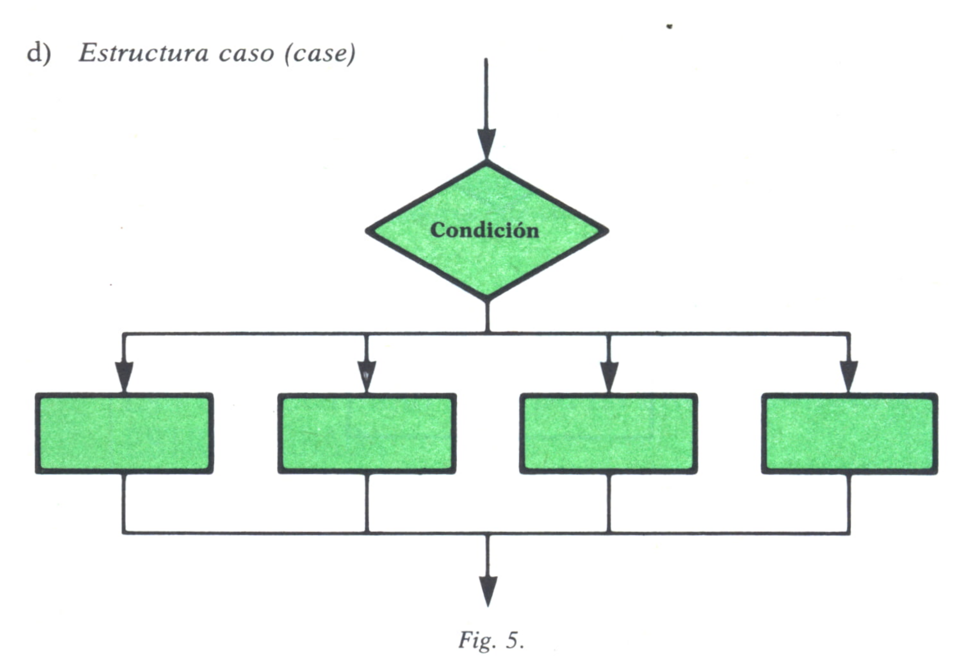


Fig. 5.

Consiste en elegir la realización de grupo de instrucciones o estructuras ordenadas entre un conjunto de éstos, dependiendo del resultado de una condición múltiple. Una condición múltiple es, por ejemplo, la que nos pregunta por el valor que hay guardado en una variable; que puede tener múltiples resultados (como veíamos en el caso de las instrucciones ON .. GOTO y ON .. GOSUB).

e) *Estructura repetitiva*

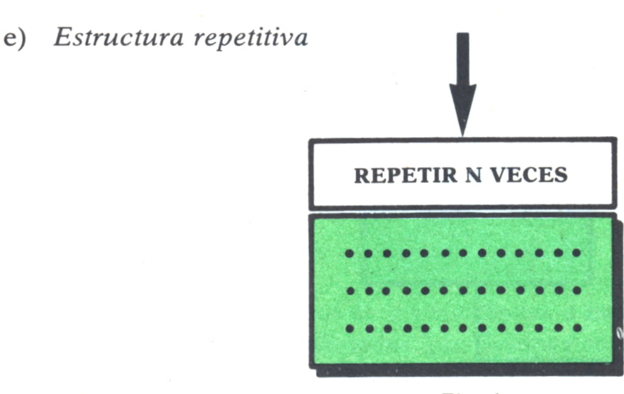


Fig. 6.

Consiste en repetir una serie de instrucciones o estructuras ordenadas un número determinado de veces (siempre en el mismo orden, por supuesto).

Cualquier programa que queramos hacer se puede poner como un conjunto de estructuras de este tipo. Veamos algunos ejemplos:

Supongamos que queremos hacer un programa (como uno que se ponía en el número anterior) que genere un número al azar entre 1 y 100 y a continuación nos vaya pidiendo números hasta que lo adivinemos; diciéndonos cada vez que metemos un número, si éste es mayor o menor que el que tenemos que adivinar. Estas operaciones se harían con las estructuras de programación estructurada que se detallan a continuación:

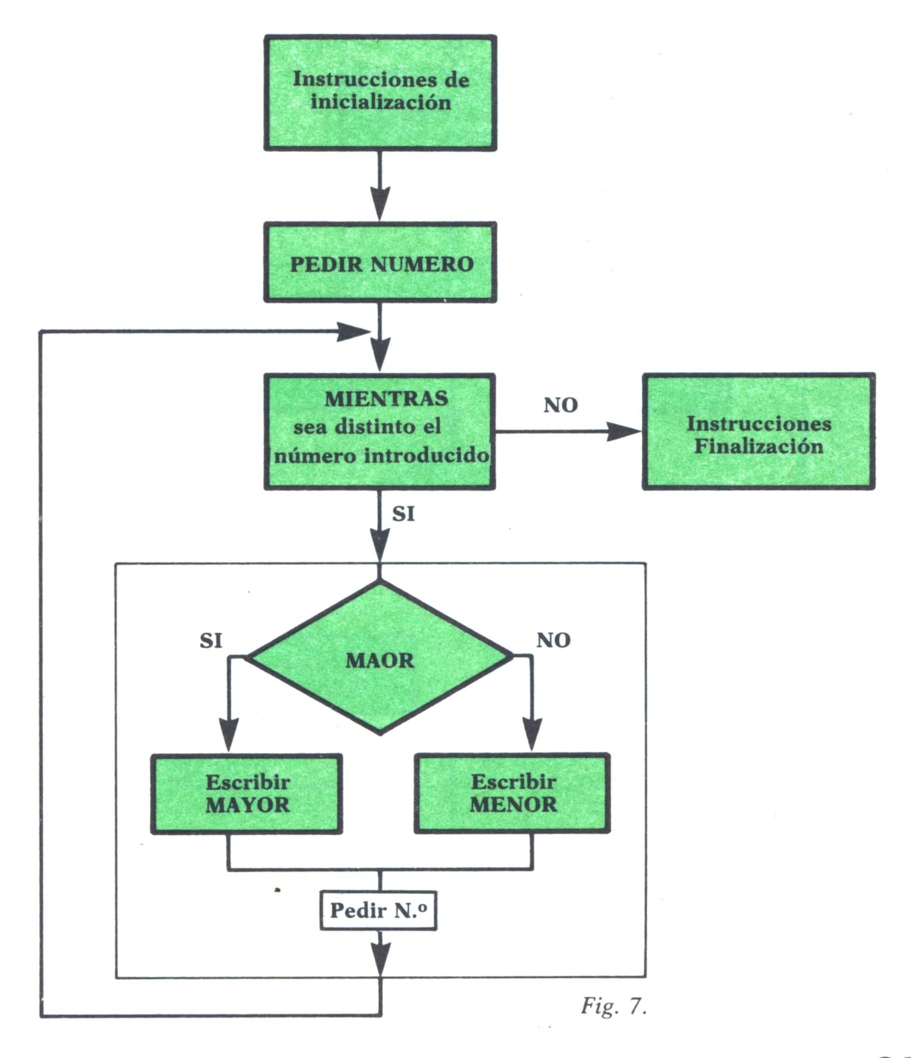


Fig. 7.

El programa para pasar de una fecha representada numéricamente a una fecha literal que se ponía en el capítulo anterior, también puede realizarse a base de una combinación de las estructuras vistas anteriormente:

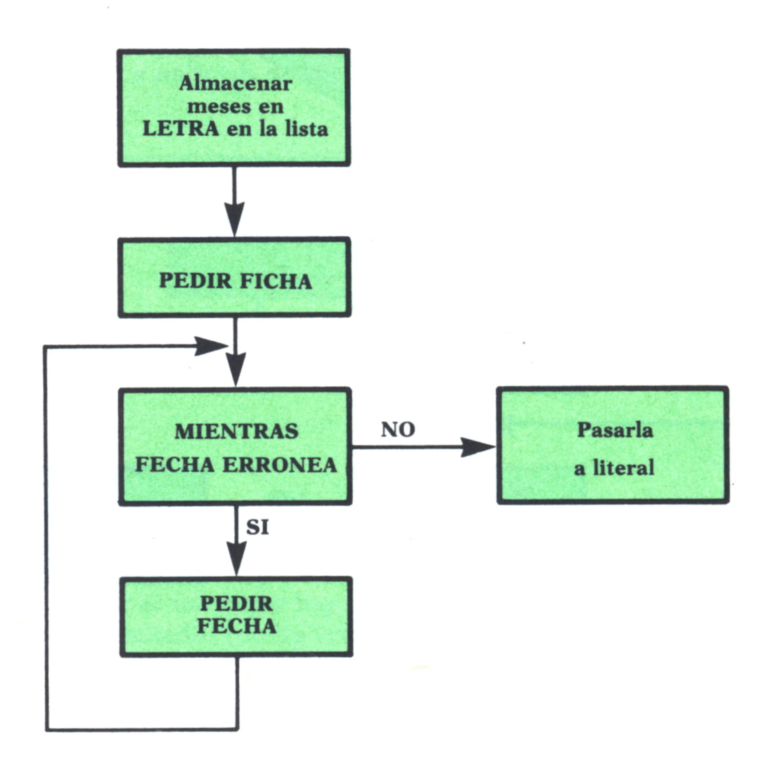


Fig. 8.

Veamos también cómo se pondría el programa para pasar un número romano a decimal:

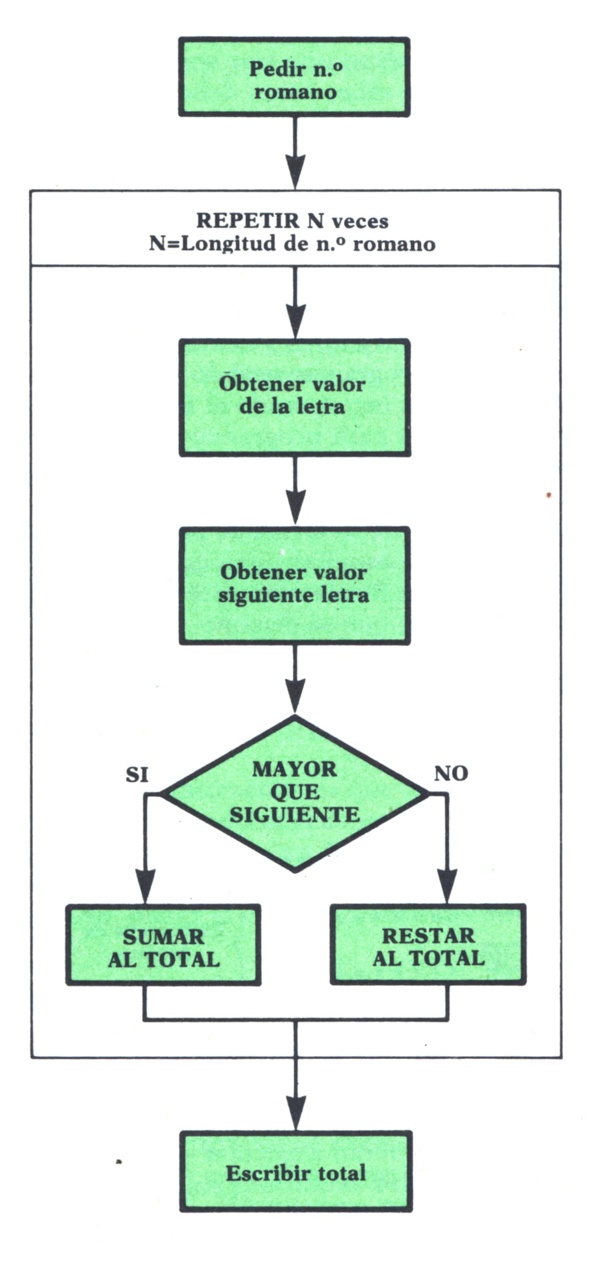


Fig. 9.

Como se habrá podido observar, la programación estructurada consiste en una serie de técnicas que nos permiten determinar las cosas que tiene que hacer un programa para realizar una operación determinada que nosotros deseemos, a base de definir éstas mediante una serie de estructuras-tipo que se deberán realizar una tras otra.

Nuestro objetivo es ahora saber cómo debemos pasar estas estructuras al lenguaje BASIC, que es el que estamos estudiando.



LAS ESTRUCTURAS EN BASIC

El BASIC es un lenguaje que existía antes de que se definieran las técnicas de la programación estructurada y por eso el BASIC estándar no permite poder pasar todas las estructuras a instrucciones del lenguaje, por lo que con él no podríamos programar usando este método. Sin embargo, hay algunos equipos que han incorporado al BASIC instrucciones propias de otros lenguajes realizados para programar con las estructuras de programación estructurada (como el PASCAL) y si permiten poder pasar estas estructuras a BASIC.

A continuación, vamos a ver la forma de poner en BASIC todas las distintas estructuras, indicando aquellas instrucciones que no existen en todos los equipos, así como la forma más aproximada de ponerlas en BASIC, con las instrucciones de que se dispone.

a) *Estructura de bloque*

Esta estructura se pondrá en BASIC colocando un conjunto de instrucciones consecutivas.

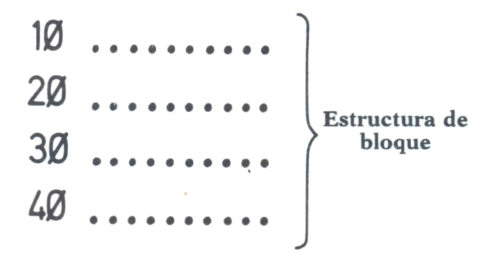


Fig. 10.

Si estas instrucciones se fueran a ejecutar en el mismo orden en varios sitios del programa deberíamos definir una subrutina. También podríamos definir una subrutina en el caso de que ello ayudase a entender mejor las partes de que consta el programa como se explicó en el capítulo anterior (como ocurría, por ejemplo, en el programa de las fechas).

b) *Estructura IF-THEN-ELSE*

Esta estructura se representa indudablemente con la instrucción IF-THEN-ELSE. En el caso de que el ordenador en el que estamos trabajando no tuviese la posibilidad ELSE, la única manera de simular esta estructura sería repitiendo de nuevo otra instrucción IF a continuación con la condición contraria.

La instrucción:

```
10 IF N MOD 2 =0 THEN PRINT "NUMERO PAR" ELSE PRINT "NUMERO IMPAR"
```

Programa 1.

habría que sustituirla por:

```
10 IF N MOD 2 =0 THEN PRINT "NUMERO PAR"
20 IF N MOD 2<>0 THEN PRINT "NUMERO IMPAR"
```

Programa 2.

Cuando el conjunto de instrucciones que queramos realizar, tanto en el THEN como en el ELSE, sea demasiado grande y no nos quepa en la instrucción a continuación de la palabra THEN o ELSE, deberemos definir una subrutina con estas instrucciones y en la instrucción IF, a continuación de THEN o ELSE, poner el GOSUB a esa subrutina.

c) *Estructura DO-WHILE*

Esta es la estructura más utilizada, ya que nos permite repetir un conjunto de instrucciones un número de veces distinto cada vez que se ejecute el programa, dependiendo de una condición que nosotros le pongamos. La utilizaremos cuando el programa no conozca de antemano cuántas veces se va a repetir. Este es el caso, por ejemplo, del programa para adivinar un número entre 1 y 100. El programa no sabe de antemano cuántas veces tendrá que repetir las operaciones de pedirte un número y compararlo porque no sabe cuánto vas a tardar en adivinar el número.



Fig. 11.

Esta estructura en BASIC se realiza mediante la instrucción WHILE-WEND que tiene el siguiente formato:

```
WHILE condición
instrucciones que se quieren repetir
..............................
..............................
WEND
```

Cada una de estas instrucciones llevarán su número correspondiente en el orden indicado. Cuando el programa llega a una instrucción WHILE, realiza lo siguiente:

a) Mira si se cumple la condición que hay a continuación de la palabra WHILE

b) Si la condición no se cumple, salta a ejecutar la instrucción que haya a continuación de WEND y continúa luego ya ejecutando las instrucciones que vengan a continuación.

c) Si la condición se cumple, pasa a ejecutar las instrucciones que haya a continuación de WHILE en su orden correspondiente hasta llegar a WEND. Cuando llega a WEND, vuelve a la WHILE y comienza de nuevo el proceso relatado en *a)*.

Veamos un programa muy sencillo como ejemplo del funcionamiento de esta instrucción:

```
10 REM MEDIA
20 REM *****
30 INPUT "TECLEA NUMERO";N
40 WHILE N<>-1
45 CON=CON+1
50 S=S+N
60 INPUT "TECLEA NUMERO";N
70 WEND
80 PRINT "LA MEDIA ES";S/CON
```

Programa 3.

Este programa nos calcula la media de una lista de números que nosotros le metamos. El número total de números no es fijo, es decir, cada vez que ejecutemos el programa querremos hallar la media de un número de valores distinto. Cuando hayamos terminado de meter los números introduciremos un –1 para indicarle al programa que ya no hay más números y que ya nos puede hallar la media. Utilizamos una instrucción WHILE para que nos pida un nuevo número y nos lo sume a la suma acumulada de los anteriores MIENTRAS N sea distinto de –1.

En el estudio de esta instrucción WHILE hay que considerar:

— Los bucles WHILE-WEND pueden ser anidados, al igual que los FOR-NEXT. En este caso se comportan también como éstos; es decir: un bucle puede contener enteramente a otro, pero no entremezclarse. Veamos este caso con el programa anterior, suponiendo que queremos que cada vez que nos pida un número, en el caso de que le metamos un número negativo nos escriba por pantalla que el número no es válido y nos pida números hasta que le metamos uno positivo:

```
10 REM MEDIA
20 REM *****
30 INPUT "TECLEA NUMERO";N
40 WHILE N<>-1
45 CON=CON+1
50 S=S+N
60 INPUT "TECLEA NUMERO";N
62 WHILE N<-1
65 INPUT "NUMERO ERRONEO.TECLEA OTRO=",N
68 WEND
70 WEND
80 PRINT "LA MEDIA ES";S/CON
```

Programa 4.

— Existe la posibilidad, no recomendada, de abandonar el bucle poniendo una instrucción GOTO dentro de las instrucciones que se repiten para que si se cumple una determinada condición se vaya fuera de éstas. No es aconsejable, ya que precisamente esta instruçcion WHILE se define para evitar el tener que usar instrucciones GOTO.

Veamos cómo este caso planteado se puede solucionar utilizando únicamente instrucciones WHILE:

```
10...
20 WHILE condición 1
30...
40 IF condición 2 GOTO 70
50...
60 WEND
70...
```

Fig. 12.

Este programa se podría poner de la forma:

```
15 IF condición 2 THEN (copia de la instrucción 30)
20 WHILE (condición 1) AND (NOT condición 2)
30... poner condición 3 verdadera
40 WHILE (NOT condición 2) AND (condición 3)
50... poner condición 3 falsa
55 WEND
57 IF condición 2 THEN (copia de la instrucción 30)
60 WEND
70...
```

Posibilidad de sustituir un caso particular de GOTO dentro de un WHILE

Fig. 13.

Esta instrucción WHILE existe únicamente en las versiones BASIC del sistema operativo MS-DOS (IBM o compatibles) y del AMSTRAD. En el resto de los ordenadores, no se puede poner la estructura DO-WHILE en BASIC, aunque si se ha realizado el esquema de lo que tiene que hacer el programa mediante estructuras de programación estructurada; este tipo de estructura se puede «simular» mediante la instrucción GOTO de la siguiente manera:

```
100 IF NOT condición THEN GOTO 210
....................
....................
200 GOTO 100
210 ...
```

Fig. 14.

También se podría hacer de esta otra forma en el caso de que quisiéramos que repitiese siempre una vez por lo menos el conjunto de instrucciones, y luego comprobase si se deben repetir más veces:

100...
....
....
200 IF condición THEN GOTO 100
210...

Fig. 15.

Veamos cómo quedaría el programa anterior en el caso de que quisiéramos ejecutarlo en un ordenador sin la instrucción WHILE, pero tratando de acercarnos lo más posible a la idea de la estructura DO-WHILE.

```
10 REM MEDIA
20 REM *****
30 INPUT "TECLEA NUMERO";N
40 IF N=-1 THEN GOTO 80
45 CON=CON+1
50 S=S+N
60 INPUT "TECLEA NUMERO";N
62 IF N>-2 THEN GOTO 70
65 INPUT "NUMERO ERRONEO.TECLEA OTRO=",N
68 GOTO 62
70 GOTO 40
80 PRINT "LA MEDIA ES";S/CON
```

Programa 5.

d) *Estructura Case*

Este tipo de estructura la pondremos en BASIC mediante la instrucción ON .. GOSUB, ya estudiada en el capítulo anterior. Las distintas subrutinas a las que se salta realizarán las operaciones que queramos hacer según el valor de la variable.

e) *Estructura repetitiva*

Esta estructura la solucionaremos en BASIC mediante una instrucción FOR-NEXT, ya de sobra conocida por los lectores.

A continuación se ofrecen los programas planteados al comienzo del capítulo realizados mediante las instrucciones vistas:

```
10 REM JUEGO PARA ADIVINAR UN NUMERO
20 REM ******************************
30 CLS
40 REM LA SIGUÍENTE INSTRUCCION ES SOLO PARA IBM
50 RANDOMIZE TIMER
60 X=INT(RND*100)+1
70 INPUT "DIME UN NUMERO";Z
```

```
80 WHILE Z <> X
90 IF X<Z THEN PRINT "ES MENOR" ELSE PRINT "ES MAYOR"
100 INPUT "DIME OTRO NUMERO DISTINTO";Z
110 WHILE (Z<0) OR (Z>500)
120 INPUT "DIME OTRO NUMERO DISTINTO";Z
130 WEND
140 WEND
150 PRINT "LO ACERTASTE"
```

Programa 6.

```
10 REM PASO DE UNA FECHA NUMERICA A TEXTO
20 REM ***********************************
25 CLS
30 DIM M$(12)
40 GOSUB 1000
42 E=1
47 WHILE E=1
48 E=0
50 INPUT "FECHA (D,M,A)";D,M,A
60 GOSUB 2000
62 REM E=1 INDICA QUE LA FECHA ES ERRONEA
65 WEND
70 PRINT "DIA:";D
80 PRINT "MES:";M$(M)
90 PRINT "AÑO:";A
100 END
1000 DATA "ENERO","FEBRERO","MARZO","ABRIL","MAYO","JUNIO","JULIO"
1010 DATA "AGOSTO","SEPTIEMBRE","OCTUBRE","NOVIEMBRE", "DICIEMBRE"
1020 FOR I=1 TO 12
1030 READ M$(I)
1040 NEXT I
1050 RETURN
2000 IF D>31 THEN PRINT "FECHA ERRONEA":E=1:RETURN
2010 IF D>28 AND M=2 THEN PRINT "FECHA ERRONEA":E=1:RETURN
2020 IF D=31 AND (M=4 OR M=6 OR M=9 OR M=11) THEN PRINT "FECHA ERRONEA":E=1:
RETURN
2030 IF M>12 THEN PRINT "FECHA ERRONEA":E=1:RETURN
2040 RETURN
```

Programa 7.

```
10 CLS
20 INPUT "DAME EL NUMERO ROMANO ";R$
30 FOR X=1 TO LEN(R$)
40 L$=MID$(R$,X,1)
50 A$=L$:GOSUB 300
60 L=A
70 S$=MID$(R$,X+1,1)
80 A$=S$:GOSUB 300
90 N=A
100 IF L>=N THEN Z=Z+L ELSE Z=Z-L
110 NEXT X
```

```
120 LOCATE 12,5:PRINT "EL RESULTADO ES ";Z
140 END
300 IF A$="M" THEN A=1000
310 IF A$="D" THEN A=500
320 IF A$="C" THEN A=100
330 IF A$="L" THEN A=50
340 IF A$="X" THEN A=10
350 IF A$="V" THEN A=5
360 IF A$="I" THEN A=1
365 IF A$="" THEN  A=0
370 RETURN
```

Programa 8.

Observamos que el programa de pasar los números romanos a decimales nos queda igual que cuando lo hicimos en el tema anterior sin saber aún la programación estructurada.



CONCLUSION

La programación estructurada es un método para realizar programas de una manera más sencilla, sobre todo cuando se trata de programas largos, en los que el hecho de dividirlos en las estructuras nos ayuda a ver mejor los pasos y las operaciones que tiene que hacer el programa.

No es, evidentemente, el único método que hay para programar y cada usurio verá a la hora de hacer un programa si lo elige o lo hace mediante saltos con instrucciones GOTO.

Debido a que el lenguaje BASIC no ha sido preparado para programar con técnicas estructuradas, hay veces que resulta más conveniente combinar las instrucciones correspondientes a éstas con alguna instrucción GOTO.

De todas formas, aunque el BASIC no está hecho para programar de manera estructurada, aquí hemos visto cómo podemos acercarnos lo más posible a las ideas de estas técnicas.

En los programas que aparezcan en este libro, aunque algunos se diseñen mediante las técnicas estructuradas, se harán sin utilizar la instrucción WHILE para lograr que puedan ser ejecutados en aquellos ordenadores que no disponen de esta instrucción.

Como final del capítulo ofrecemos un programa que simula una carrera de caballos en pantalla utilizando primero instrucciones GOTO y después instrucciones WHILE-WEND.

```
10 REM CARRERA DE CABALLOS
20 REM ******************
30 RANDOMIZE TIMER
40 CLS
50 LOCATE 1,1 :PRINT 1 :A=3
60 LOCATE 2,1 :PRINT 2 :B=3
70 LET CA=INT(RND* 2)+1
75 REM EN CA SE SELECCIONA EL CABALLO QUE AVANZA
80 IF CA=1 THEN A=A+1:LOCATE 1,A:PRINT "*"
90 IF CA=2 THEN B=B+1:LOCATE 2,B:PRINT "*"
100 IF (A<80) AND (B<80) THEN GOTO 70
110 LOCATE 4,3
120 PRINT "HA GANADO EL ";CA
```

Programa 9.

```
10 REM CARRERA DE CABALLOS
20 REM ******************
30 RANDOMIZE TIMER
40 CLS
50 LOCATE 1,1 :PRINT 1 :A=3
60 LOCATE 2,1 :PRINT 2 :B=3
65 WHILE A<80 AND B<80
70 LET CA=INT(RND* 2)+1
75 REM EN CA SE SELECCIONA EL CABALLO QUE AVANZA
80 IF CA=1 THEN A=A+1:LOCATE 1,A:PRINT "*"
90 IF CA=2 THEN B=B+1:LOCATE 2,B:PRINT "*"
100 WEND
110 LOCATE 4,3
120 PRINT "HA GANADO EL ";CA
```

Programa 10.

EJERCICIOS:

1. Modifica el programa de la carrera de caballos para que corra un número cualquiera de caballos que nos pida previamente el programa. Utilizar una lista (DIM) para guardar la columna por la que va cada caballo.

2. Hacer un programa que calcule el número de días que hay entre dos fechas dadas. Realizarlo primero con instrucciones GOTO y luego con WHILE-WEND.

3. Intenta realizar cualquier programa que tengas hecho con instrucciones GOTO con instrucciones WHILE-WEND únicamente.

RESPUESTAS:

2.

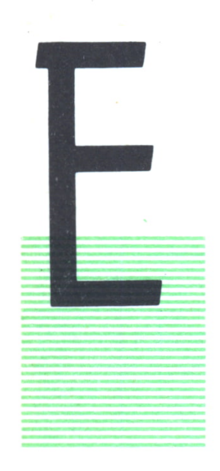
```
1 REM DIAS ENTRE DOS FECHAS USANDO GOTO
2 REM ********************************
10 CLS
20 INPUT "dime la PRIMERA FECHA";A,B,C
30 INPUT "dime la SEGUNDA FECHA";H,E,F
35 REM COMPUTO DE AÑOS INTERMEDIOS
40 IF(B<E) OR ( B=E AND H>A) THEN D=(F-C)*365 ELSE D=((F-C)-1)*365
50 IF B=E AND A<H THEN D=D+(H-A):GOTO  240
55 REM COMPUTO DE LOS MESES QUE NO FORMAN UN AÑO COMPLETO
60 CON=B
70 IF CON<12 THEN CON=CON+1 ELSE CON=1
80 IF CON=E THEN GOTO 130
90 IF CON=2 THEN D=D+28
100 IF CON=1 OR CON=3 OR CON=5 OR CON=7 OR CON=8 OR CON=10 OR CON=12 THEN D=D+31
110 IF CON=4 OR CON=6 OR CON=9 OR CON=11 THEN D=D+30
120 GOTO 70
125 REM COMPUTO DE LOS DIAS QUE NO FORMAN UN MES COMPLETO
130 D=D+H
140 IF B=2 THEN D=D+(28-A)
150 IF B=1 OR B=3 OR B=5 OR B=7 OR B=8 OR B=10 OR B=12 THEN D=D+(31-A)
160 IF B=4 OR B=6 OR B=9 OR B=11 THEN D=D+(30-A)
165 REM COMPUTO DE LOS AÑOS BISIESTOS
170 FOR I=C+1 TO F-1
180 IF I/4= INT(I/4) THEN D=D+1
190 NEXT I
200 IF F=C AND B<3 AND E>2 AND F/4 = INT(F/4) THEN D=D+1
210 IF F<>C AND B<3 AND C/4= INT(C/4) THEN D=D+1
220 IF F<>C AND E>2 AND F/4= INT(F/4) THEN D=D+1
240 PRINT "LOS DIAS SON:";D
```

Programa 11.

RESOLUCION DE ALGUNOS PROBLEMAS TIPICOS EN BASIC 4



INTRODUCCION



EN este capítulo vamos a ver cómo se podrían resolver mediante un programa BASIC dos problemas típicos que se nos pueden plantear en muchas ocasiones. El primero de ellos será el de ordenar una serie de números que tengamos guardados en una lista y el segundo el de encontrar el camino más corto entre dos ciudades incluidas dentro de una red de ciudades.



COMO ORDENAR UN CONJUNTO DE DATOS

Supongamos que tenemos una lista de N elementos:

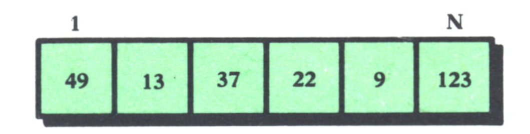


Fig. 1.

y deseamos ordenarlos de mayor a menor.

Se nos pueden plantear dos casos distintos: que haya suficiente espacio en memoria como para poder utilizar otra lista para ir metiendo los elementos ordenados tal como vemos en la figura o que el número N de ele-

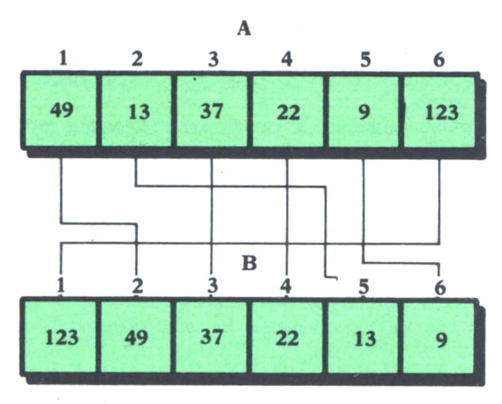


Fig. 2.

mentos de la lista sea tan grande que no tengamos espacio en memoria para declarar otra lista distinta y tengamos que ordenar los elementos sobre la misma lista. Los métodos que veamos para este segundo caso los podemos aplicar de todas formas siempre, ya que aunque haya espacio suficiente en memoria para declarar otra lista, hay muchas veces que nos interesa que el programa nos deje los valores ordenados en la MISMA lista o nos resulta incómodo tener que utilizar otra lista, pudiendo aplicar los programas que nos dejan los valores en la MISMA lista.

a) *Ordenamiento con dos listas*

La idea que hay que realizar es la expresada en la figura anterior: buscar el mayor de los elementos y guardarlo en la primera posición de la segunda lista, luego buscar el mayor de los que queden y guardarlo en la segunda posición de la segunda lista, y así sucesivamente.

El siguiente programa realiza este caso:

```
10 INPUT "numero de elementos";N
20 DIM A(N): DIM B(N)
30 FOR K=1 TO N
40 PRINT "ELEMENTO N.";K
50 INPUT A(K)
60 NEXT K
70 GOSUB 1000
80 FOR K=1 TO N
```

```
90 PRINT B(K);
100 NEXT K
110 END
1000 REM SUBRUTINA PARA ORDENAR LA LISTA A(N) DEJANDO EL RESULTADO EN B(N)
1010 REM ***************************************************************
1015 REM Esta subrutina debe estar en un programa en el que queramos
1016 REM ordenar los elementos guardados en A(N)
1020 FOR I=1 TO N
1030 GOSUB 2000
1040 B(I)=A(MAX)
1050 A(MAX)=0
1055 NEXT I
1060 RETURN
2000 REM Esta subrutina halla el mayor de los elementos de la lista A
2010 MAX=1
2020 FOR J=2 TO N
2030 IF A(J)>A(MAX) THEN MAX=J
2040 NEXT J
2050 REM EN MAX DEJA EL LUGAR DONDE ESTA EL MAYOR
2060 RETURN
```

Programa 1.

Cuando encontramos el mayor de los números de la primera lista, colocamos en su lugar un cero para luego buscar la siguiente vez el mayor de los números que resten.

b) *Ordenamiento sobre la misma lista*

Vamos a ver dos métodos para este caso:

— El primer método consistiría en buscar el mayor de los elementos de la lista y colocarlo en la primera posición de ésta después de haber desplazado un lugar hacia la derecha los elementos comprendidos entre el primer lugar y aquel en el que se encontraba el mayor. Luego tendríamos que repetir la misma operación para todos los elementos de la lista, excepto el primero, y así seguiríamos sucesivamente hasta llegar al último, con lo que habremos conseguido ordenar todos los elementos.

La siguiente figura ilustra este método:

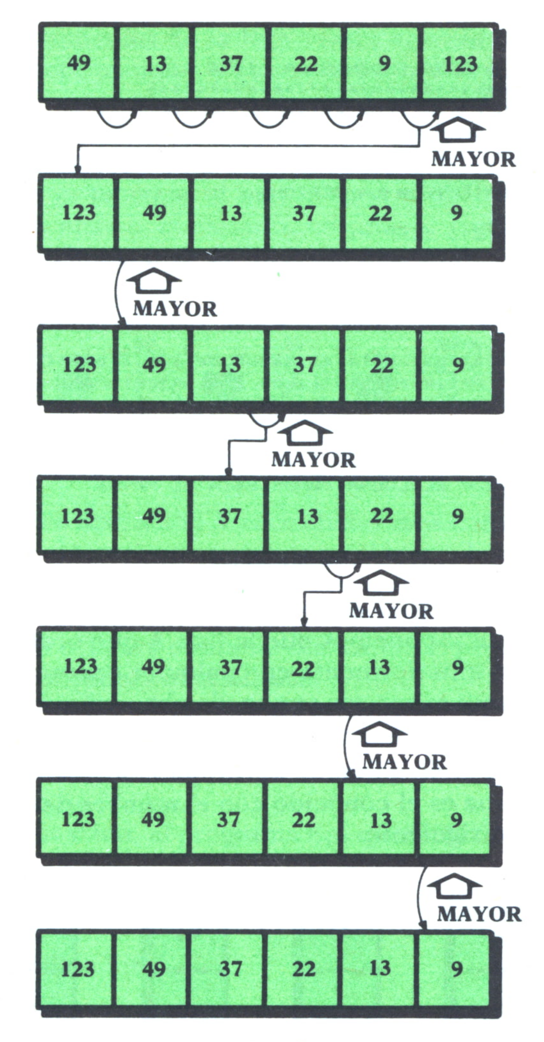


Fig. 3.

El programa BASIC que realiza este método es el siguiente:

```
10 INPUT "numero de elementos";N
20 DIM A(N)
30 FOR K=1 TO N
40 PRINT "ELEMENTO N.";K
50 INPUT A(K)
60 NEXT K
70 GOSUB 1000
80 FOR K=1 TO N
90 PRINT A(K);
100 NEXT K
```

```
110 END
1000 REM SUBRUTINA PARA ORDENAR LA LISTA A(N)
1010 REM ***********************************
1020 REM Esta subrutina debe estar en un programa en el que queramos
1030 REM ordenar los elementos guardados en A(N).
1040 FOR I=1 TO N
1050 GOSUB 2000
1055 REM DESPLAZAMIENTO A LA DERECHA
1057 M=A(MAX)
1060 FOR J=MAX TO I+1 STEP -1
1070 A(J)=A(J-1)
1080 NEXT J
1090 A(I)=M
1110 NEXT I
1120 RETURN
2000 REM Esta subrutina halla el mayor de los elementos de la lista A
2010 REM a partir del elemento I.Los I-1 anteriores ya están ordenados
2020 MAX=I
2030 FOR J=I+1 TO N
2040 IF A(J)>A(MAX) THEN MAX=J
2050 NEXT J
2060 REM EN MAX DEJA EL LUGAR DONDE ESTA EL MAYOR
2070 RETURN
```

Programa 2.

Hay que tener en cuenta que todos estos programas son partes de un programa en los que suponemos que la lista A que ordenamos ha podido obtener sus valores anteriormente en cualquier operación del programa. En los programas aquí expuestos se pone la obtención de estos valores mediante INPUT a modo de ejemplo.

— El segundo método que vamos a ver para ordenar los valores de una lista sobre ella misma es el conocido con el nombre de método de la burbuja. Consiste en lo siguiente:

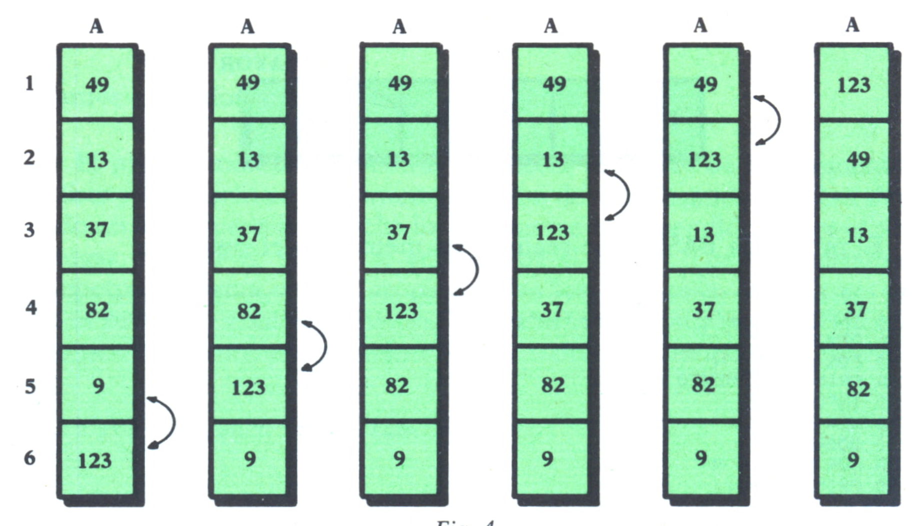


Fig. 4.

Si entre los elementos de la lista vamos comparando dos a dos (cada uno con el que tiene encima), y ponemos encima el mayor de los dos; después de haber comparado todos, empezando por el último, habremos conseguido que el mayor de todos haya quedado el primero. En la figura 4 se puede observar este proceso.

Si ahora, que ya tenemos el mayor en el primer lugar de la lista, volvemos a hacer lo mismo, conseguiremos que el segundo más grande nos quede en el segundo lugar de la lista; como vemos en la figura:

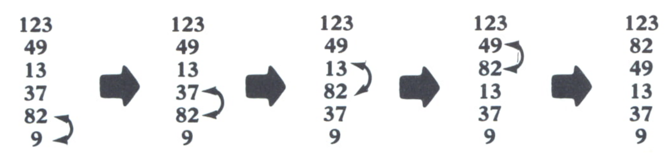


Fig. 5.

Repitiendo esta operación *N* veces habremos conseguido que la tercera vez nos quede el tercero más grande en el tercer lugar de la lista, la cuarta vez el cuarto y así sucesivamente hasta que al repetirlo las *N* veces habrá quedado la lista ordenada. Se deja como ejercicio al lector, que a partir del ejemplo de la figura continúe realizando el proceso las cuatro veces restantes para comprobar el resultado.

Este método recibe el nombre de método de la burbuja, porque cada vez que se repiten todas las comparaciones el número mayor «sube» hasta su lugar correspondiente como si lo hiciera en una burbuja que ascendiera desde el fondo de la lista.

Con estas ideas vistas, el programa nos quedaría de la siguiente manera:

```
10 INPUT "numero de elementos";N
20 DIM A(N)
30 FOR K=1 TO N
40 PRINT "ELEMENTO N.";K
50 INPUT A(K)
60 NEXT K
70 GOSUB 1000
80 FOR K=1 TO N
90 PRINT A(K);
100 NEXT K
110 END
1000 REM SUBRUTINA PARA ORDENAR LA LISTA A(N) POR EL METODO DE LA BURBUJA
1010 REM ************************************************************
1020 REM Esta subrutina debe estar en un programa en el que queramos
1030 REM ordenar los elementos guardados en A(N)
1040 FOR I=1 TO N
1050 REM COMPARACION PAREJA A PAREJA
1060 FOR J=N TO I+1 STEP -1
1070 IF A(J)>A(J-1) THEN M=A(J-1):A(J-1)=A(J):A(J)=M
1080 NEXT J
1090 NEXT I
1100 RETURN
```

Programa 3.

Aunque en todos estos ejemplos hemos ordenado listas numéricas, de la misma manera podríamos ordenar palabras guardadas en una lista alfanumérica. Las palabras nos quedarían ordenadas en orden alfabético de mayor a menor, es decir, la primera sería la que empezara por la letra que estuviera más al final en el abecedario. Recordar que cuando comparamos dos palabras, es mayor aquélla que se encontrara después si las ordenásemos por orden alfabético.

A continuación se ofrece este último programa realizado mediante el método de la burbuja, para el caso de ordenar datos alfanuméricos.

```
10 INPUT "numero de PALABRAS ";N
20 DIM A$(N)
30 FOR K=1 TO N
40 PRINT "ELEMENTO N.";K
50 INPUT A$(K)
60 NEXT K
70 GOSUB 1000
75 PRINT
80 FOR K=1 TO N
90 PRINT A$(K)
100 NEXT K
110 END
1000 REM SUBRUTINA PARA ORDENAR LA LISTA A$(N) POR EL METODO DE LA BURBUJA
1010 REM ***************************************************************
1020 REM Esta subrutina debe estar en un programa en el que queramos
1030 REM ordenar los elementos guardados en A$(N)
1040 FOR I=1 TO N
1050 REM COMPARACION PAREJA A PAREJA
1060 FOR J=N TO I+1 STEP -1
1070 IF A$(J)>A$(J-1) THEN M$=A$(J-1):A$(J-1)=A$(J):A$(J)=M$
1080 NEXT J
1090 NEXT I
1100 RETURN
```

Programa 4.

Si los datos alfanuméricos estuvieran compuestos por caracteres distintos de letras, serían mayores aquellos caracteres que tuvieran un código ASCII mayor.



EL PROBLEMA DEL CAMINO MAS CORTO

Otro problema típico que nos puede surgir en muchos tipos de programas es el de encontrar el camino más corto entre dos ciudades incluidas dentro de una red de ciudades. El problema se puede generalizar al camino más corto entre dos puntos de una red.

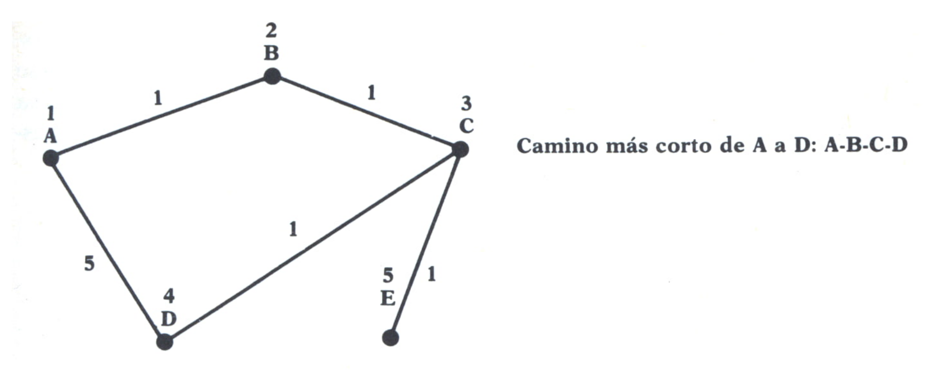


Fig. 6.

Supongamos para ello que la red tiene *N* ciudades y que tenemos una tabla C como la de la figura en la que tenemos almacenadas las distancias entre las ciudades que tienen una carretera directa entre ellas:

C

	1	2	3	4	5
1	0	1	MAX	5	MAX
2	1	0	1	MAX	MAX
3	MAX	1	0	1	1
4	5	MAX	1	0	MAX
5	MAX	MAX	1	MAX	0

Fig. 7.

Esta figura representaría la tabla correspondiente a la red de la figura anterior. En aquellas variables correspondientes a la distancia entre dos ciudades que no tienen camino directo entre ellas se guarda un valor que vamos a llamar MAX y que corresponderá con un valor muy alto. Esto se hace así porque el método que vamos a desarrollar y que luego explicamos, así lo requiere.

A continuación se ofrece el programa que resuelve este problema. Este programa pide por pantalla los nombres de las ciudades y a continuación las distancias entre aquellas ciudades de la red que estén unidas por una carretera directa (cuando no haya carretera directa entre las ciudades entre las que se pide la distancia bastará con pulsar RETURN o ENTER).

El programa a continuación halla los caminos más cortos entre las distintas ciudades de la red aplicando el algoritmo de Floyd (que después se explicará someramente su idea) y después pide al usuario por pantalla una pareja de ciudades entre las que se quiera saber el camino más corto, escribiendo éste y su distancia.

```
10 REM      ************************************************************
20 REM      **                                                        **
30 REM      **    PROGRAMA PARA HALLAR EL CAMINO MAS CORTO ENTRE DOS  **
40 REM      **    CIUDADES INCLUIDAS DENTRO DE UNA RED DE CARRETERAS  **
50 REM      **                                                        **
70 REM      **                                                        **
80 REM      ************************************************************
90 CLS
100 INPUT "INTRODUCE EL NUMERO DE CIUDADES QUE FORMAN LA RED";N
110 DIM C(N,N):DIM D(N,N):DIM N$(N)
120 LET MAX=1E+10
130 FOR I=1 TO N
140 PRINT
150 PRINT "TECLEA EL NOMBRE DE LA CIUDAD Nº";I
160 INPUT N$(I)
170 FOR J=1 TO N :LET C(I,J)=MAX:NEXT J
180 LET C(I,I)=0
190 PRINT
200 NEXT I
210 CLS
220 PRINT "INTRODUCE A CONTINUACION LA DISTANCIA ENTRE LAS CIUDADES QUE SE";
230 PRINT "INDICAN CUANDO ENTRE ELLAS HAYA UN CAMINO DIRECTO.";
240 PRINT "SI NO HAY CAMINO DIRECTO PULSA SIMPLEMENTE <RETURN>";
250 PRINT :PRINT
260 FOR I=1 TO N
270 FOR J=I+1 TO N
280 PRINT N$(I);"-";N$(J);
290 INPUT DIS
300 IF DIS=0 THEN GOTO 320
310 LET C(I,J)=DIS:LET C(J,I)=DIS
320 NEXT J:NEXT I
325 PRINT :PRINT "!UN MOMENTO!,ESTOY CALCULANDO"
330 FOR I=1 TO N
340 FOR J=1 TO N
350 LET D(I,J)=I
360 NEXT J:NEXT I
370 FOR I=1 TO N
380 FOR K=1 TO N
390 IF I=K OR C(I,K)=MAX THEN GOTO 440
400 FOR J=1 TO N
410 IF J=K OR C(K,J)=MAX THEN 430
420 IF C(I,K)+C(K,J)<C(I,J) THEN LET C(I,J)=C(I,K)+C(K,J) :LET D(I,J)=D(K,J)
430 NEXT J
440 NEXT K:NEXT I
450 GOSUB 1000
460 PRINT :PRINT
470 INPUT "INTRODUCE LOS NUMEROS DE DOS CIUDADES (SEPARADOS POR COMAS) ENTRE
          LAS QUE QUIERAS SABER EL CAMINO MAS CORTO.";C1,C2
480 PRINT :PRINT N$(C1);" - ";N$(C2):PRINT
490 LET DIS=C(C1,C2)
500 PRINT N$(C1);" - ";
510 IF C1=C2 THEN PRINT :PRINT :GOTO 535
520 LET C1=D(C2,C1)
530 GOTO 500
535 PRINT "la distancia minima es";DIS
540 INPUT "QUIERES TERMINAR(S/N)";R$
550 IF R$="N" THEN GOTO 450
560 IF R$="S" THEN END
570 GOTO 540
1000 CLS
1010 FOR I=1 TO INT(N/2)
1020 PRINT (I*2)-1;N$((I*2)-1);TAB(40);I*2;N$(I*2)
1030 NEXT I
1040 IF N/2<>INT(N/2) THEN PRINT N;N$(N)
```

```
1050 RETURN
1060 REM   ****************************************************************
1070 REM   **  MODIFICACIONES PARA SPECTRUM:                             **
1080 REM   **  110 DIM C(N,N):DIM D(N,N):DIM N$(N,10)                    **
1090 REM   **  560 Cambiar END por GOTO 9999                             **
1100 REM   **  MODIFICACIONES PARA COMMODORE:                            **
1110 REM   **  Cambiar CLS POR PRINT CHR$(147) en 90,210 y 1000          **
1120 REM   ****************************************************************
```

Programa 5.

Vamos a dar seguidamente las ideas fundamentales en que se basa el algoritmo de Floyd que utiliza el programa para resolver el problema:

En primer lugar, se crea una nueva tabla que se llama D y que tiene el siguiente significado: en D(i,j) se va a guardar la penúltima ciudad del camino más corto para ir de *i* a *j*.

Inicialmente el método considera que la penúltima ciudad del camino más corto para ir de *i* a *j* es *i*, ya que aún no hemos hallado ningún otro camino para ir de *i* a *j* que el camino directo si lo hay.

A continuación el algoritmo estudia si para cada pareja de ciudades de la red existe otra ciudad intermedia unida a ambas tal que yendo a través de ella el camino entre ambas resulte más corto que la distancia que tenemos en la tabla C para ir de una a otra (de ahí que cuando entre dos ciudades no había camino directo, poníamos un valor muy alto como distancia entre ellas: para que se cumpliera siempre la condición de que yendo por una ciudad intermedia, el camino sea más corto). Cuando ocurra esto, el programa colocará en el lugar correspondiente a la distancia entre dos ciudades de la tabla C la nueva distancia; y como penúltima ciudad del camino más corto para ir de una a otra la penúltima ciudad del camino más corto para ir de la ciudad intermedia a la ciudad destino.

Despues de repetir esto para cada pareja de vértices, al final tendremos en la tabla C las distancias más cortas entre todas las ciudades de la red, y en la tabla D la penúltima ciudad del camino más corto entre una y otra.

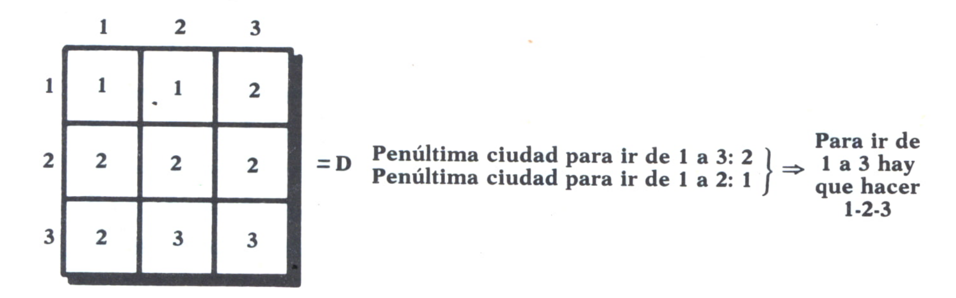


Fig. 8.

Estas operaciones se hacen entre las líneas 370 y 440 del programa.

Una vez que tenemos las tablas de esta forma, para obtener el camino más corto entre dos ciudades de la red, bastará con obtener primero la penúltima ciudad del camino más corto para ir de una a otra. Luego a continuación, la penúltima ciudad del camino más corto para ir de la ciudad origen a la penúltima ciudad obtenida anteriormente y así sucesivamente hasta que obtengamos que la penútima ciudad es ya la ciudad origen, tal y como se explica en la figura anterior.

Estas operaciones las hace el programa entre las líneas 500 y 530, considerando C2 como la ciudad origen y C1 como la ciudad destino, para que el camino (que le vamos obteniendo a partir de la penúltima ciudad) nos salga en el orden pedido.

Veamos a continuación con un sencillísimo ejemplo, cómo funcionaría este método explicado que realiza el programa anterior. Supongamos que tenemos la red que pusimos anteriormente:

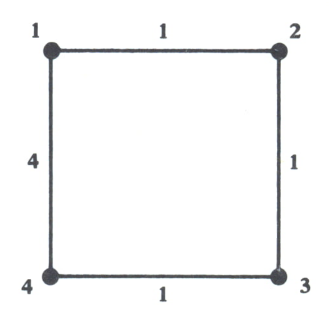


Fig. 9.

Inicialmente las tablas C y D valdrían:

C

0	1	MAX	4
1	0	1	MAX
MAX	1	0	1
4	MAX	1	0

D

1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

Fig. 10.

Al recorrer el programa todas las posibles parejas de ciudades, encontrará que entre las ciudades 1 y 3 existe una ciudad intermedia (la 2), tal

que la distancia para ir de 1 a 3 es menor pasando por 2 (1+1=2) que la que tenemos puesta hasta ahora en la tabla C (que era MAX por no haber camino directo entre ellas). De esta forma, las tablas C y D quedarían con los siguientes valores:

C

0	1	2	4
1	0	1	MAX
2	1	0	1
4	MAX	1	0

D

1	1	2	1
2	2	2	2
2	3	3	3
4	4	4	4

Se ha incluido ya el caso de ir de 3 a 1 que aunque el programa lo encontraría más tarde, sería igual

Fig. 11.

A continuación encontrará que para las ciudades 1 y 4 la ciudad 3 es una ciudad intermedia que cumple que yendo primero de 1 a 3 (cuya distancia es 2, encontrada anteriormente) y luego de 3 a 4 (cuya distancia es 1) es más corto que la distancia que tenemos en C para ir de 1 a 4 (el camino directo). Después de esto, las tablas nos quedarían ahora:

C

0	1	2	3
1	0	1	MAX
2	1	0	1
3	MAX	1	0

D

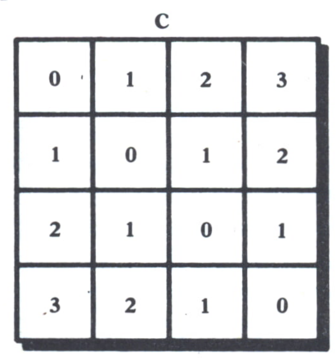
1	1	2	3
2	2	2	2
2	3	3	3
2	4	4	4

Como en el anterior, se ha incluido el caso de ir de 4 a 1

Fig. 12.

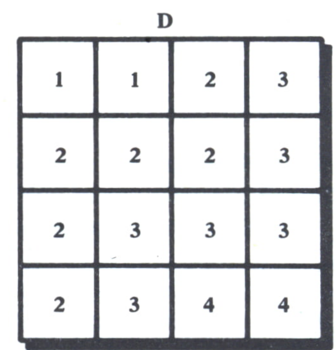
Del resto de las parejas de vértices que analiza, únicamente encontrará ya que para ir de 2 a 4 hay una ciudad intermedia (la 3), cuyo camino es

más corto que el que tenemos en C para ir de 2 a 4 (MAX). Así, las tablas nos quedarían finalmente:



C

0	1	2	3
1	0	1	2
2	1	0	1
3	2	1	0



D

1	1	2	3
2	2	2	3
2	3	3	3
2	3	4	4

Fig. 13.

y los caminos más cortos entre dos ciudades cualesquiera se hallarían tal y como se dijo anteriormente.

En las operaciones de las instrucciones 370-440, las variables I y J corresponden a la pareja de ciudades entre la que estamos viendo si hay un camino más corto yendo por una ciudad intermedia, y la variable K a las ciudades intermedias con las que vamos probando.

Aunque el método resulta un poco complicado, esperamos que el lector haya sacado una idea general de las bases de éste, y que el programa sea útil para resolver cualquier tipo de problema en el que haya que saber el camino más corto entre dos puntos incluidos dentro de una red.

EJERCICIOS:

1. Cambiar los programas de ordenar los elementos de una lista de mayor a menor para que los ordene de menor a mayor.

2. Hacer un programa que pida por pantalla el número de habitantes de una serie de ciudades y ordene éstas de mayor a menor número de habitantes.

3. Hacer un programa que escriba la lista de una clase. Los nombres los leerá de un DATA en el orden en el que se los introduzcan y los escribirá, uno por línea, ordenados por orden alfabético y con su número delante.

LOS GRAFICOS EN BASIC: DIBUJOS UTILIZANDO CARACTERES 5

INTRODUCCION



NA de las cosas que resulta más atractiva a la hora de programar es el saber cómo hacer gráficos y dibujos en pantalla mediante un programa. Como ya se dedicó un número entero de la coleeción a éstos, en estos próximos capítulos no se va a profundizar en todas las posibilidades existentes, ya que sería repetir todo lo que se expuso en ese número, sino que se van a tratar de dar las líneas maestras de la programación con gráficos estudiando las instrucciones mas importantes.



LOS CARACTERES: PRIMERA FORMA DE EMPEZAR A DIBUJAR

Con lo que se ha visto hasta ahora, el lector probablemente ya habrá podido realizar algunos programas para hacer dibujos sencillos combinando las instrucciones PRINT y LOCATE, y las funciones TAB y SPACE$ (o SPC) cuando se necesiten.

Así, una línea se puede dibujar con asteriscos ò puntos, mediante el siguiente programa sencillo:

****

Fig. 1.

```
10 REM ASTERISCOS
20 REM **********
25 REM En todos los programas está puesto el LOCATE para MS-DOS.
26 REM Recordar variantes para otros equipos
30 CLS
40 FOR COL=1 TO 50
50 LOCATE 3,COL:PRINT "*"
60 NEXT COL
```

Programa 1.

Para dibujar una línea diagonal bastaría con hacer variar al mismo tiempo la fila y la columna:

```
10 REM ASTERISCOS
20 REM **********
30 CLS
40 FOR COL=1 TO 20
50 LOCATE COL,COL:PRINT "*"
60 NEXT COL
```

Programa 2.

La combinación de estas líneas nos permitiría la realización de cualquier tipo de figuras. Así, a continuación podemos observar tres programas que nos harían los dibujos de las tres figuras que les acompañan.

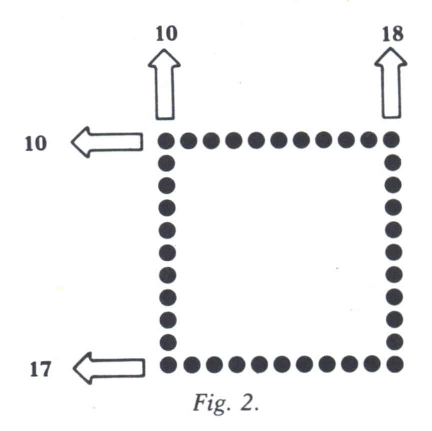


Fig. 2.

```
10 REM CUADRADO
20 REM ********
30 CLS
40 FOR COL=10 TO 18
50 LOCATE 10,COL:PRINT "."
60 NEXT COL
70 FOR COL=10 TO 18
80 LOCATE 17,COL:PRINT "."
90 NEXT COL
100 FOR FIL=10 TO 17
110 LOCATE FIL,10:PRINT "."
120 NEXT FIL
130 FOR FIL=10 TO 17
140 LOCATE FIL,18:PRINT "."
150 NEXT FIL
```

Programa 3.

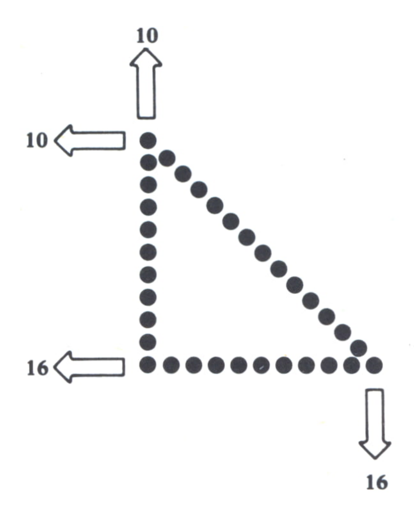


Fig. 3.

```
10 REM TRIANGULO
20 REM *********
25 CLS
30 FOR FIL=10 TO 16
40 LOCATE FIL,10:PRINT "."
50 NEXT FIL
60 FOR COL=10 TO 16
70 LOCATE 16,COL:PRINT "."
80  NEXT COL
90 FOR A=10 TO 16
100 LOCATE A,A:PRINT "."
110 NEXT A
```

Programa 4.

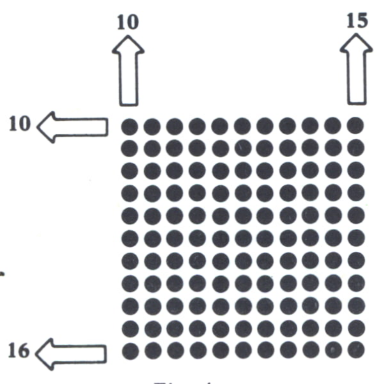


Fig. 4.

```
10 REM RECTANGULO DE PUNTOS
20 REM ********************
25 CLS
30 FOR I=10 TO 16
40 FOR J=10 TO 15
50 LOCATE I,J :PRINT "."
60 NEXT J
70 NEXT I
```

Programa 5.

Notar cómo se pueden dibujar tanto el contorno de las figuras como se ha hecho en los dos primeros casos como la figura también rellenada por dentro como en el tercer programa.

Sin embargo, tenemos que recordar que el conjunto de caracteres que se pueden escribir dentro de una instrucción PRINT no son únicamente aquellos que se pueden introducir mediante una tecla directa, sino que cada ordenador tiene 256 caracteres numerados, mediante lo que se denomina código ASCII, del 0 al 255. Estos caracteres son aquellos que se corresponden con las teclas (letras mayúsculas, minúsculas y acentuadas y caracteres como los asteriscos, puntos, signos aritméticos y demás) y un conjunto de caracteres que corresponde cada uno a un cierto gráfico. Recordar que para que un programa nos escribiera estos últimos por pantalla debemos utilizar la función CHR$(N). N es un número comprendido entre 0 y 255, y la función obtiene como resultado el carácter cuyo código ASCII es N.

Así, podemos escribir, mediante un programa, cualquiera de los caracteres ASCII, escribiendo la instrucción:

PRINT CHR$(N)

estando en el lugar de N, el número correspondiente al carácter que queramos dibujar, o la variable donde se encuentre guardado dicho número.

Para saber los caracteres correspondientes a cada uno de los códigos ASCII en nuestro ordenador, bastará con que ejecutemos el siguiente programa (que se puso en el número anterior al explicar la función CHR$), o consultar el manual de nuestro ordenador en el que suelen venir todos los caracteres con su código ASCII.

```
10 REM CODIGOS ASCII
20 REM *************
30 CLS
40 FOR I=0 TO 255
50 PRINT I;"-->"; CHR$(I)
60 NEXT I
```

Programa 6.

Veamos a continuación cómo podemos realizar dibujos un poco más «vistosos» utilizando este conjunto de caracteres más «especiales».

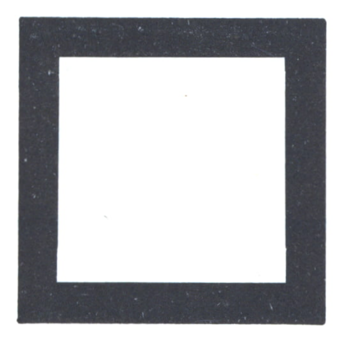


Fig. 5.

Un cuadrado de este estilo podríamos dibujarlo en nuestro ordenador utilizando los caracteres de la forma:

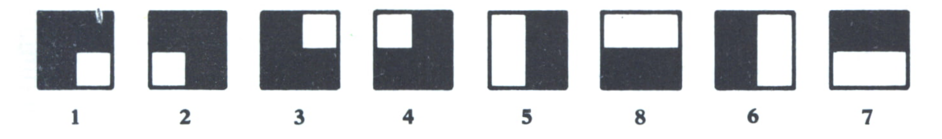


Fig. 6.

que se encuentran en todos los ordenadores, aunque no tienen el mismo código en todos ellos ni la misma forma gráfica. Es importante aquí recordar que mientras en todos los ordenadores las letras y números poseen el mismo código ASCII, los caracteres gráficos especiales ni poseen el mismo código ASCII ni son iguales para todos, habiendo algunos propios en cada equipo.

Para este sencillo ejemplo del cuadrado, vamos a utilizar los siguientes caracteres ASCII de los distintos ordenadores:

CARACTERES	SEPCTRUM	AMSTRAD	COMMODORE
1	139	135	111
2	135	139	112
3	142	141	108
4	141	142	186
5 6	133-138	133-138	167-116
8 7	140 131	140 131	114 163

Fig. 7.

El programa nos quedaría de la siguiente manera:

```
10 REM CUADRADO
20 REM ********
30 DIM C(8)
40 FOR I=1 TO 8
50 PRINT "INTRODUCE CODIGO ASCIII DEL CARACTER NUMERO";I
60 INPUT C(I)
70 NEXT I
75 CLS
80 PRINT CHR$(C(1));
90 FOR I=1 TO 3
100 PRINT CHR$(C(7));
110 NEXT I
120 PRINT CHR$(C(2))
130 FOR I=1 TO 3
140 PRINT CHR$(C(6));"    ";CHR$(C(5))
150 NEXT I
160 PRINT CHR$(C(3));
170 FOR I=1 TO 3
180 PRINT CHR$(C(8));
190 NEXT I
200 PRINT CHR$(C(4));
```

Programa 7.



LAS FIGURAS EN MOVIMIENTO

Vamos a ver en este apartado una técnica muy sencilla que nos va a permitir aumentar en gran manera las posibilidades de nuestros dibujos en el ordenador.

En numerosos programas resulta muy interesante conseguir que una determinada figura se mueva por la pantalla. La técnica general que podemos aplicar a cada uno de los casos es la siguiente: *Dibujar la figura que queramos que se mueva en un lugar determinado de la pantalla y a continuación borrarla, dibujándola en la posición de al lado de la anterior, según la dirección en la que queremos que se mueva, volviendo a repetir a continuación la acción de borrarla y dibujarla al lado.* La realización de este proceso un número determinado de veces produce, al realizarse en la pantalla, el efecto óptico de que la figura se está moviendo.



Fig. 8.

Veamos una primera aplicación de esto en el siguiente programa que realiza el movimiento de un asterisco a lo largo de la pantalla en sentido horizontal:

```
10 REM ASTERISCOS
20 REM **********
30 CLS
40 FOR COL=1 TO 50
50 LOCATE 3,COL:PRINT "*"
55 LOCATE 3,COL:PRINT " "
60 NEXT COL
```

Programa 8.

El programa se ha hecho suponiendo que la pantalla consta de 80 columnas. En aquellos ordenadores cuya pantalla tenga menos columnas, habrá que poner como límite superior de la instrucción FOR el número de columnas de ésta.



TAMBIEN PODEMOS GRADUAR LA VELOCIDAD

Al ejecutar el programa anterior, probablemente le haya parecido al usuario que el asterisco avanzaba por la pantalla a una velocidad excesivamente rápida. Veamos cómo podemos conseguir que la figura que queremos mover lo haga a la velocidad que deseemos.

Observando el programa anterior se puede ver que cada vez que se repite la instrucción **FOR** se dibuja y se borra el asterisco en la posición correspondiente. Para conseguir que se mueva más despacio, lo que deberemos hacer será que el programa después de dibujar el asterisco y antes de borrarle se quede un momento «parado». Esto se consigue colocando, entre medias de las instrucciones que realizan ambas operaciones, la instrucción:

FOR K=1 TO 100 :NEXT K

Cuando el programa llegue a ejecutar esta instrucción, dará a K el valor 1, a continuación se encontrará con la instrucción **NEXT** que asignará a K el siguiente valor (2); y, como todavía es menor que 100, continuará ejecutando la instrucción FOR, realizando el mismo proceso hasta que K llegue a 100, momento en el que terminará de ejecutar esta instrucción. Como se puede observar entonces, el efecto de esta instrucción es el de provocar un retardo, ya que mientras el programa está realizando las operaciones reseñadas correspondientes a ella, a efectos de lo que ve el usuario, el programa está parado sin hacer nada.

El tiempo de este retardo se puede graduar muy fácilmente, cambiando el límite superior de la instrucción **FOR**. Cuanto mayor pongamos este número, más tiempo estará «parado» el programa y cuanto menor sea, menos tiempo estará «parado».

Veamos cómo quedaría el programa del movimiento del asterisco si queremos que lo haga a menor velocidad:

```
10 REM ASTERISCOS
20 REM **********
30 CLS
40 FOR COL=1 TO 50
50 LOCATE 3,COL:PRINT "*"
53 FOR K=1 TO 100 :NEXT K
55 LOCATE 3,COL:PRINT " "
60 NEXT COL
```

Programa 9.

La velocidad con la que queramos que se mueva el asterisco podemos

determinarla muy fácilmente aumentando o disminuyendo el retardo según queramos que vaya más despacio o más de prisa, respectivamente.

FOR K=1 TO 200 :NEXT K

haría que disminuyera la velocidad y:

FOR K=1 TO 50: NEXT K

haría que aumentara la velocidad.

Es importante notar en estos programas en los que se utiliza esta instrucción de retardo, que la variable que se pone en el FOR debe ser una variable que no se exista en el programa para otra cosa.

El movimiento podría realizarse también en vertical o diagonal. A continuación se ofrece un programa que mueve en diagonal un muñequito que existe en los códigos ASCII del MS-DOS y del AMSTRAD. Para los otros equipos se pide el código ASCII del carácter que se quiere mover, ya que no existe ningún carácter con la forma del muñequito.

```
10 REM FIGURA EN DIAGONAL
20 REM ******************
25 CLS
30 PRINT "AMSTRAD            -->1"
40 PRINT "IBM O COMPATIBLES -->2"
50 PRINT "OTRO EQUIPO        -->3"
60 INPUT " TECLEA EL NUMERO QUE CORRESPONDA ",N
70 ON N GOSUB 1000,2000,3000
75 CLS
80 FOR I=1  TO 20
90 LOCATE I,I:PRINT CHR$(M)
100 FOR K= 1 TO 90 : NEXT K
110 LOCATE I,I:PRINT " "
120 NEXT I
130 END
1000 M=250:RETURN
2000 M=2:RETURN
3000 INPUT " INTRODUCE CODIGO DEL CARACTER QUE QUIERES MOVER";M
3010 RETURN
```

Programa 10.

Hasta ahora hemos visto los casos en los que queremos mover un solo carácter. Supongamos que en vez de un carácter queremos mover una figura que hayamos hecho como combinación de una serie de caracteres, como, por ejemplo, el cuadrado que dibujamos anteriormente:

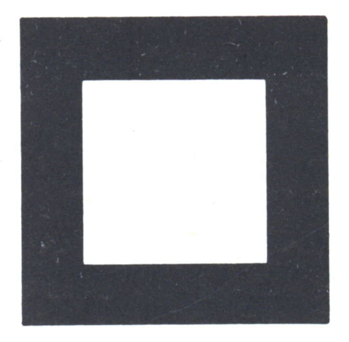


Fig. 9.

La técnica para conseguir el movimiento es la misma: dibujarlo en una posición, borrarlo y dibujarlo en la posición de al lado. El problema nos surge a la hora de borrarlo, ya que ahora no lo podemos hacer tan sencillo como lo hacíamos antes que poníamos:

PRINT " ".

El problema ahora, que aún no hemos visto las posibilidades que ofrece el uso de los colores, debemos solucionarlo de la siguiente forma:

— Debemos definir una subrutina para dibujar el cuadrado. Esta subrutina utilizará la lista representada en la figura para tener los ocho caracteres que se necesitan para dibujar el cuadrado.

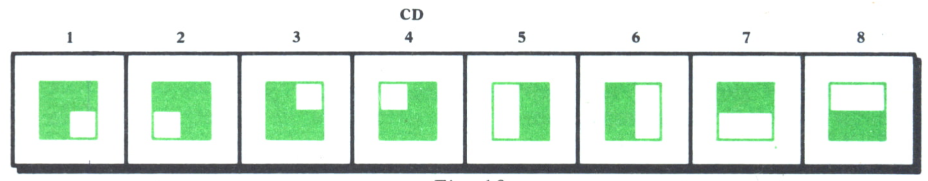


Fig. 10.

Antes de llamar a la subrutina (con la instrucción GOSUB) se deberán guardar en la lista los caracteres correspondientes. A continuación vemos cómo quedaría la subrutina:

```
1000 REM SUBRUTINA PARA DIBUJAR EL CUADRADO
1010 REM P:COLUMNA DE INICIO
1020 REM Los LOCATE están puestos para el AMSTRAD
1030 CLS
1050 LOCATE P,10
1060 PRINT C$(1) ;
1070 FOR I=1 TO 3
1080 PRINT C$(7);
1090 NEXT I
```

```
1100 PRINT C$(2)
1110 FOR I=1 TO 3
1120 LOCATE P, 10-I
1130 PRINT C$(6);"    ";C$(5)
1140 NEXT I
1150 LOCATE P,6
1160 PRINT C$(3);
1170 FOR I=1 TO 3
1180 PRINT C$(8);
1190 NEXT I
1200 PRINT C$(4);
1210 RETURN
```

Programa 11.

– Ahora ya resulta muy fácil la operación de borrar el cuadrado, ya que bastará con que, cuando queramos borrarle, llamemos a la subrutina habiendo guardado previamente en todas las variables de la lista el valor: « ».

El programa para mover el cuadrado nos quedaría entonces de la siguiente manera:

```
10 REM CUADRADO MOVIL
20 REM **************
25 DIM A$(8): DIM C$(8)
30 CLS
40 FOR I=1 TO 8
50 PRINT " INTRODUCE CODIGO DEL CARACTER";I
60 INPUT X
65 REM TAMBIEN SE PUEDEN COLOCAR LOS CODIGOS DE LOS CARACTERES EN UN DATA.
66 REM Y LAS DOS INSTRUCCIONES ANTERIORES SE SUSTITUIRIAN POR UN READ(X)
70 A$(I)=CHR$(X)
80 NEXT I
90 FOR P=1 TO 30 STEP 3
100 FOR J=1 TO 8
110 C$(J)=A$(J)
120 NEXT J
130 GOSUB 1000
140 FOR K=1 TO 75:NEXT K
150 FOR J=1 TO 8
160 C$(J)=" "
170 NEXT J
180 NEXT P
190 END
1000 REM SUBRUTINA PARA DIBUJAR EL CUADRADO
1010 REM P:COLUMNA DE INICIO
1020 REM Los LOCATE están puestos para el AMSTRAD
1030 CLS
1050 LOCATE P,10
1060 PRINT C$(1) ;
1070 FOR I=1 TO 3
1080 PRINT C$(7);
1090 NEXT I
1100 PRINT C$(2)
1110 FOR I=1 TO 3
```

```
1120 LOCATE P, 10-I
1130 PRINT C$(6);"    ";C$(5)
1140 NEXT I
1150 LOCATE P,6
1160 PRINT C$(3);
1170 FOR I=1 TO 3
1180 PRINT C$(8);
1190 NEXT I
1200 PRINT C$(4);
1210 RETURN
```

Programa 12.

La variable P indica la posición en la que se empieza a dibujar el cuadrado cada vez. Dependiendo del **STEP** que pongamos en su **FOR**, el cuadrado «saltará» más o menos.



EL MOVIMIENTO CON TECLAS

Vamos a ver a continuación otra posibilidad de movimiento de figuras mediante un programa: es el caso de cuando queremos que una figura se mueva hacia un lado u otro cuando pulsemos unas determinadas teclas. La técnica para conseguir esto es la misma que vimos anteriormente: para mover una figura hacia la derecha cuando se haya pulsado una determinada tecla, lo que deberemos hacer será borrar la figura de donde la tenemos dibujada y dibujarla en la posición que se encuentre inmediatamente a su derecha. Lo mismo se haría para el caso de que el movimiento fuese hacia la izquierda, hacia arriba o hacia abajo.

Veamos cómo nos quedaría un programa para mover un carácter cualquiera con cuatro teclas:

```
10 REM PROGRAMA PARA MOVER UN CARACTER CON 4 TECLAS
20 REM ******************************************
30 CLS
40 F=5:C=10
50 LOCATE F,C :PRINT "*"
60 A$=INKEY$
70 IF A$="" THEN GOTO 60
80 REM TECLAS PARA MOVERSE: U,J,H,N
90 REM ****************************
100 REM F PARA TERMINAR
110 IF A$="F" THEN END
120 LOCATE F,C:PRINT " "
130 IF A$="U" THEN IF F>1 THEN F=F-1
140 IF A$="H" THEN IF C>1 THEN C=C-1
150 IF A$="J" THEN C=C+1
```

```
160 IF A$="N"' THEN F=F+1
180  LOCATE F,C :PRINT "*"
190 GOTO 60
```

Programa 13.

Todas las técnicas de movimiento explicadas en estos tres últimos apartados son aplicables a cualquier tipo de figura que dibujemos en nuestro ordenador mediante un programa, tanto las que hemos visto que podemos hacer hasta ahora como las veamos en próximos capítulos.



LOS COLORES EN EL ORDENADOR

Vamos a ver en este apartado, de una manera rápida, cómo funciona el tema de los colores en los distintos ordenadores.

Como idea general, hay que decir que cuando se posea un monitor **MONOCOLOR**, los colores que podemos utilizar en un programa son los **MISMOS**; aunque éstos nos saldrán en la pantalla como distintas tonalidades e intensidades del color del monitor.

a) *Los colores en el SPECTRUM*

El **SPECTRUM** posee ocho colores diferentes numerados del 0 al 7, tal y como se indica en la figura adjunta:



Fig. 11.

Cuando encendemos el ordenador y ejecutamos cualquier programa sin poner ningún tipo de color, el color con el que nos escribe o dibuja lo que le mandemos en el programa es el blanco (código 7) y el color del fondo de la pantalla es el negro (código 0).

Para cambiar el color con el que queramos que nos escriba o dibuje debemos utilizar la instrucción INK. Poniendo:

INK N

siendo N un número entre 0 y 7 o una variable en la que tenemos guardado dicho número. A partir del momento en el que el ordenador ejecute esta instrucción y hasta que vuelva a ejecutar otra instrucción INK o apaguemos el ordenador, TODAS las cosas que se escriban en pantalla se harán con el color, cuyo código se ha puesto en la instrucción INK. Así, si ejecutamos el programa:

```
10 INK 2
20 PRINT "HOLA"
```

Programa 14.

escribirá HOLA en rojo y a continuación si mandamos listar, o introducimos nuevas instrucciones, escribirá ya todo en rojo.

Para cambiar el color del fondo de la pantalla se utiliza la instrucción PAPER de la misma manera que la INK. Cuando ponemos:

PAPER N

nos cambia el color del fondo de la pantalla a aquel cuyo código es N. Es importante notar aquí que el color del fondo no lo cambiará cuando ejecute la instrucción PAPER, sino cuando escriba algo en pantalla o se borre la pantalla después de haber ejecutado la PAPER.

El siguiente programa nos obtiene todos los posibles colores de fondo de la pantalla:

```
10 REM COLORES DE PANTALLA-SPECTRUM
20 REM ****************************
30 FOR A=1 TO 7
40 PAPER A
50 CLS
60 FOR K=1 TO 500:NEXT K
70 NEXT A
```

Programa 15.

También podemos cambiar el color de los bordes de la pantalla. Esto se consigue mediante la instrucción BORDER seguida de un número, que será, lógicamente, el del código de color correspondiente.

Las instrucciones INK y PAPER pueden incluirse también dentro de un PRINT, de este modo, el cambio de color de la tinta o de la pantalla sólo afectará a los datos que estén dentro de ese PRINT.

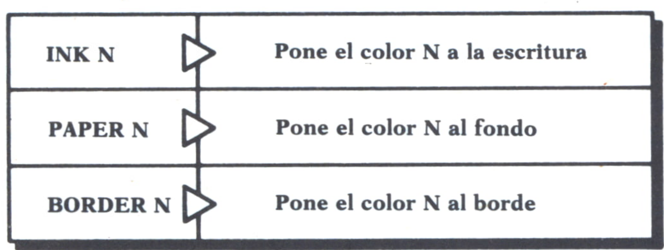


Fig. 12.

b) *Los colores en el AMSTRAD*

En el AMSTRAD pueden llegarse a conseguir hasta 27 colores diferentes. Cada color posee un código numérico que es el siguiente:

0	Negro	13	Blanco
1	Azul	14	Azul pastel
2	Azul brillante	15	Naranja
3	Rojo	16	Rosa
4	Magenta	17	Magenta pastel
5	Malva	18	Verde brillante
6	Rojo brillante	19	Verde marino
7	Púrpura	20	Cian brillante
8	Magenta brillante	21	Verde lima
9	Verde	22	Verde pastel
10	Ciano	23	Ciano pastel
11	Azul cielo	24	Amarillo brillante
12	Amarillo	25	Amarillo pastel

26	Blanco brillante

Fig. 13.

El AMSTRAD utiliza el concepto de «tintero». Tiene internamente 16 «tinteros» (numerados del 0 al 15). Cada uno de estos «tinteros» pueden tener un color diferente, y cuando queramos cambiar el color del fondo de la pantalla o de la escritura en ésta, deberemos asignarles un «tintero», siendo a partir de ese momento el color de la pantalla o de la escritura el que tenga el «tintero» que le hemos asignado. Existen, por tanto, instrucciones para asignar colores a cada uno de los 16 «tinteros» y asignar «tinteros» a la pantalla o a la escritura, que vamos a ver a continuación:

— INK N,M: Asigna al tintero N, el color cuyo código es M
— PEN N: Asigna a la escritura en pantalla el tintero N
— PAPER N: Asigna al fondo de pantalla el tintero N

También hay que tener en cuenta en el AMSTRAD que estando en MODO 1 únicamente se pueden asignar los cuatro primeros tinteros (del 0 al 3) y estando en modo 2 sólo los dos primeros (el 0 y el 1).

Veamos a continuación un programa que mueve un muñequito, cambiándolo de color cada vez que se mueve:

```
10 REM MUÑECO DE COLORES
20 REM *****************
25 MODE 0
30 FOR J=1 TO 15
40 INK J,J
50 NEXT J
60 CLS
70 FOR I=1  TO 15
80 PEN I
90 LOCATE I,I:PRINT CHR$(250)
100 FOR K= 1 TO 90 : NEXT K
110 LOCATE I,I:PRINT " "
120 NEXT I
130 FOR J=1 TO  5
140 INK J,J+15
150 NEXT J
160 FOR I=16 TO 20
170 PEN I-15
180 LOCATE I,I:PRINT CHR$(250)
190 FOR K= 1 TO 90 : NEXT K
200 LOCATE I,I:PRINT " "
210 NEXT I
```

Programa 16.

La instrucción BORDER también la tiene el AMSTRAD, y funciona como en el SPECTRUM:

— BORDER N: Cambia el color de los bordes de la pantalla a aquel cuyo código es N.

c) *Los colores en el COMMODORE*

El Commodore dispone de 16 colores, tanto para la escritura como para el fondo de la pantalla.

Para variar los colores de la escritura se hace mediante unos códigos ASCII, cuyo significado es ése. Así, si ponemos:

PRINT CHR$(N)

siendo N el código ASCII correspondiente a alguno de los colores, nos escribirá a partir de ese momento con ese color.

En el manual del equipo aparecen los códigos correspondientes a cada color.

Por otra parte, la pantalla y el borde también se pueden cambiar de color. Los 16 colores para éstos son los siguientes:

0	**Negro**
1	**Blanco**
2	**Rojo**
3	**Cian**
4	**Púrpura**
5	**Verde**
6	**Azul**
7	**Amarillo**

8	**Naranja**
9	**Marrón**
10	**Rojo claro**
11	**Gris 1**
12	**Gris 2**
13	**Verde claro**
14	**Azul claro**
15	**Gris 3**

Fig. 14.

Para cambiar el color del fondo de la pantalla, se deberá poner:

POKE 53281,N, siendo N el código del color correspondiente.

Para cambiar el color del borde, pondremos:

POKE 53280,N, siendo N el código del color correspondiente.

d) *Los colores en IBM*

El color de la pantalla y de la escritura se selecciona mediante la instrucción COLOR. Esta instrucción tiene dos formatos:

a) Cuando se está en modo texto, es decir, lo que se escriba o dibuje

se haga mediante una instrucción PRINT; la instrucción COLOR tendrá el siguiente formato:

— COLOR carácter, fondo

En el lugar de «carácter» se deberá colocar el número del color con el que queramos que nos escriba los caracteres y en el de «fondo» aquél que queramos poner como fondo. Se aconseja consultar el manual del ordenador correspondiente para ver la tabla de los códigos correspondientes a cada color para cada uno de los casos, ya que puede variar algo de unos equipos a otros. También sería conveniente consultar el manual, si en algún ordenador esta instrucción no diese los resultados apetecidos, ya que no es estándar para todos los IBM y puede haber en algunos que tenga otros parámetros o cambie alguno de los expuestos.

b) Cuando se va a dibujar mediante las instrucciones de gráficos que se verán más adelante:

— COLOR fondo, paleta

Donde pone «fondo» se podrá poner un valor entre 0 y 15 que seleccionará el color del fondo de la pantalla.

Donde pone «paleta» habrá que poner un valor entre 0 y 1. Este valor indica las dos posibles gamas de colores que existen, de acuerdo con el siguiente cuadro:

Color	Paleta 0	Paleta 1
0	Color del fondo	Color del fondo
1	Verde	Cyan
2	Rojo	Magenta
3	Marrón	Blanco

Fig. 15.

En las instrucciones para hacer gráficos de IBM, que veremos más adelante, siempre hay que especificar el color, con un número del 0 al 3, que corresponderá a un color u otro dependiendo de que hayamos puesto un 0 o un 1 en la paleta y de acuerdo con la tabla anterior.

Así, después de haber puesto la instrucción:

COLOR 3,0

si mandamos dibujar algo mediante alguna de las instrucciones de gráficos, que veremos en próximos capítulos con el color 2, nos lo dibujará en rojo, ya que al haber especificado la paleta 0, el color 2 de la paleta 0 es el rojo.

Cuando estamos en el modo para hacer gráficos, que ya veremos más tarde cómo se entra en él, también se pueden escribir caracteres mediante PRINT y podemos decidir el color con el que queremos escribirlos añadiendo a la instrucción COLOR, en este último formato visto, algún parámetro que dependerá de cada equipo y para lo que se aconseja consultar el manual del ordenador en cada caso.

Jugando con estas instrucciones de colores que hemos visto en este apartado, podemos conseguir realizar las operaciones de borrado de las figuras para simular movimiento de una manera más sencilla de como lo hicimos anteriormente. BORRAR una figura será dibujarla habiendo puesto como color de escritura el MISMO DEL FONDO DE LA PANTALLA.

Veamos esta idea para el programa de mover un cuadrado en el SPECTRUM.

```
10 REM CUADRADO MOVIL
20 REM **************
30 DIM C$(8,1)
40 CLS
50 FOR I=1 TO 8
60 PRINT " INTRODUCE CODIGO DEL CARACTER";I
70 INPUT X
80 REM TAMBIEN SE PUEDEN COLOCAR LOS CODIGOS DE LOS CARACTERES EN UN DATA.
90 REM Y LAS DOS INSTRUCCIONES ANTERIORES SE SUSTITUIRIAN POR UN READ(X)
100 LET C$(I,1)=CHR$(X)
110 NEXT I
120 FOR P=1 TO 30 STEP 3
130 INK 5
140 GOSUB 1000
150 FOR K=1 TO 75:NEXT K
160 REM PARA BORRAR SE PONE EL MISMO COLOR DEL FONDO
170 INK 0
180 GOSUB 1000
190 NEXT P
200 GOTO 9999
1000 REM SUBRUTINA PARA DIBUJAR EL CUADRADO
1010 REM P:COLUMNA DE INICIO
1020 CLS
1030 PRINT AT 10,P ;
1040 PRINT C$(1) ;
1050 FOR I=1 TO 3
1060 PRINT C$(7);
1070 NEXT I
1080 PRINT C$(2)
1090 FOR I=1 TO 3
1100 PRINT AT 10-I,P;
1110 PRINT C$(6);"   ";C$(5)
1120 NEXT I
1130 PRINT AT 6,P;
```

```
1140 PRINT C$(3);
1150 FOR I=1 TO 3
1160 PRINT C$(8);
1170 NEXT I
1180 PRINT C$(4);
1190 RETURN
```

Programa 17.

Este programa se podría trasladar a cualquiera de los otros ordenadores, cambiando simplemente los códigos de los caracteres por sus códigos correspondientes y las operaciones de poner como color de escritura el mismo del fondo de la pantalla por las que se correspondan de acuerdo con lo explicado en este último apartado.

El cuadro que se ofrece a continuación nos indica el número de filas y columnas que tiene cada ordenador en modo texto, es decir para situarse mediante la instrucción LOCATE o equivalentes (PRINT AT en el SPECTRUM) y escribir mediante instrucciones PRINT.

	FILAS	COLUMNAS
SPECTRUM	**22**	**32**
AMSTRAD	**25**	**20 — MODO 0** **40 — MODO 1** **80 — MODO 2**
COMMODORE	**25**	**40**
MS-DOS	**24**	**80 (40 en SCREEN 1)**

Las filas se cuentas en éstos de abajo a arriba y la numeración comienza en 0

Los modos del AMSTRAD y SCREEN en MS-DOS se tratan en el capítulo 7

Fig. 16.

EJERCICIOS:

1. Realizar un programa que pida los valores de los puntos que tienen cada uno de los equipos de uno de los grupos de la liga de fútbol, baloncesto o cualquier otro deporte y dibuje en pantalla la clasificación de la siguiente manera:



Fig. 17.

2. Hacer un programa que escriba una franja horizontal, formada por una serie de cuadrados, cada uno de uno de los distintos colores que posea nuestro ordenador.

3. Realizar un programa que haga un dibujo geométrico a base de puntos combinando distintos colores

4. Hacer un programa que mueva un carácter a lo largo de la pantalla hasta que se pulse una tecla y se le pare.

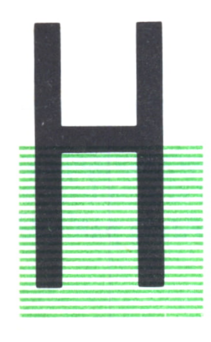
RESPUESTAS:

1.

```
10 REM *********************************************
20 REM EN EL PRIMER BLOQUE DEL PROGRAMA HABRIA QUE
30 REM PEDIR LOS NOMBRES Y LOS PUNTOS DE CADA EQUIPO.
40 REM SUPONEMOS QUE LOS NOMBRES SE GUARDAN EN N$(N)
50 REM Y LOS PUNTOS EN P(N) Y QUE EN ESTAS LISTAS ESTAN
60 REM ORDENADOS DE MAYOR A MENOR PUNTUACION.
70 REM **********************************************
80 REM DIBUJO DE LA FIGURA
90 REM *******************
95 INPUT "CODIGO DEL CARACTER CUYA FORMA ES UN CUADRADO";COD
100 CLS
110 FOR I=1 TO N
120 LOCATE I,1:PRINT N$(I)
130 LOCATE I,20
140 FOR J=1 TO P(I)
150 PRINT CHR$(COD) ;
160 NEXT J
170 NEXT I
```

Programa 18.

DISEÑANDO NUESTROS PROPIOS CARACTERES 6



HAY algunos ordenadores como el SPECTRUM y el AMSTRAD que nos permiten diseñar caracteres con la forma que nosotros queramos para luego posteriormente dibujarlos en pantalla.



METODO GENERAL DE DISEÑO DE ESTOS CARACTERES

Estos caracteres que nosotros diseñemos ocuparán en pantalla, cuando sean escritos, el espacio que normalmente ocupa un carácter, es decir, un pequeño cuadrado.

Para diseñar estos caracteres, se supone que este cuadrado está dividido en una malla de 8 por 8, tal y como se indica en la figura:

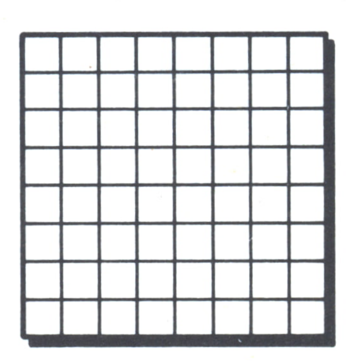


Fig. 1.

En esta malla se deben rellenar los pequeños cuadritos de la manera que queramos para que nos configuren la forma del dibujo que queremos

diseñar. En la siguiente figura vemos una manera de rellenar los cuadritos de la malla para conseguir la forma de una nave espacial:

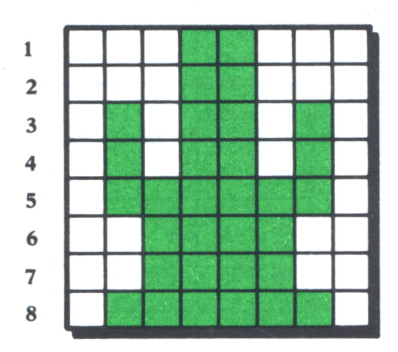


Fig. 2.

Una vez que tenemos realizada esta operación, debemos coger las ocho filas de la malla una a una y con cada una hay que hacer lo siguiente: por cada cuadrito de la fila que hayamos rellenado debemos poner un 1 y por cada cuadrito de la fila que hayamos dejado en blanco sin rellenar un 0. De esta manera por cada fila tendremos una lista de unos y ceros de 8 elementos.

Veamos para el ejemplo anterior de la nave espacial cómo tendríamos que hacer esto:

— La primera fila es la siguiente:

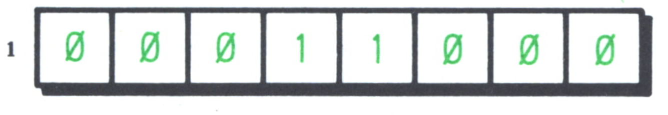


Fig. 3.

y, por tanto, la lista de unos y ceros que le corresponde será:



Fig. 4.

Realizando la misma operación para el resto de las filas nos quedaría lo siguiente:

Fila 1	0	0	0	1	1	0	0	0
Fila 2	0	0	0	1	1	0	0	0
Fila 3	0	1	0	1	1	0	1	0
Fila 4	0	1	0	1	1	0	1	0
Fila 5	0	1	1	1	1	1	1	0
Fila 6	0	0	1	1	1	1	0	0
Fila 7	0	0	1	1	1	1	0	0
Fila 8	0	1	1	1	1	1	1	0

Fig. 5.

Una vez que tenemos realizado esto para la figura que queremos diseñar, tendremos que introducirlo en el ordenador según se explica a continuación.



DEFINICION DE CARACTERES EN EL SPECTRUM

El funcionamiento en el SPECTRUM de estos caracteres nuevos que definimos nosotros es el siguiente: el carácter que nosotros hemos diseñado es asociado a una tecla del teclado del ordenador. Luego, estando en modo gráfico (AL QUE SE ENTRA PULSANDO SIMULTANEAMENTE LAS TECLAS CAPS SHIFT Y GRAPHICS) cada vez que se pulse esa tecla saldrá la forma del carácter que hayamos diseñado.

Así, si hubiéramos asociado el carácter diseñado anteriormente de la nave espacial a la tecla A, cada vez que pulsemos en modo gráfico la tecla A nos escribirá en la pantalla:



Fig. 6.

El problema que nos queda por resolver ahora es cómo decirle a nuestro ordenador que la figura que hemos diseñado la queremos asociar a la tecla A. Esto se hace mediante las siguientes ocho instrucciones de programa:

```
10 POKE USR "A" +0,BIN 00011000
20 POKE USR "A" +1,BIN 00011000
30 POKE USR "A" +2,BIN 01011010
40 POKE USR "A" +3,BIN 01011010
50 POKE USR "A" +4,BIN 01111110
60 POKE USR "A" +5,BIN 00111100
70 POKE USR "A" +6,BIN 00111100
80 POKE USR "A" +7,BIN 01111110
```

Programa 1.

Las ocho instrucciones se deben colocar consecutivas. Después de la palabra USR y entre comillas va la tecla a la que queremos asignar nuestro nuevo carácter, y en la **PRIMERA** instrucción (la que lleva +0 después de la tecla entre comillas) se pone después de la palabra **BIN** la lista de ceros y unos correspondiente a la **PRIMERA** fila de la malla del dibujo diseñado. La **SEGUNDA** instrucción llevará la lista de unos y ceros correspondiente a la **SEGUNDA** fila, y así sucesivamente. Puede observarse que el programa puesto anteriormente correspondería a la definición de la nave espacial diseñada al principio asociándola a la tecla **A**.

Una vez que un programa ha ejecutado estas instrucciones la tecla correspondiente se queda con el nuevo carácter asociado para modo gráfico hasta que se le asocie otro carácter mediante otro conjunto de instrucciones de la misma forma o se apague el ordenador.

A continuación se ofrece un programa para mover por pantalla la nave espacial que hemos diseñado con dos teclas:

```
10 REM PROGRAMA PARA MOVER UN CARACTER CON 2 TECLAS
20 REM ********************************************
30 CLS
40 F=5:C=10
50  PRINT AT F,C;"♟"
60 A$=INKEY$
70 IF A$="" THEN GOTO 60
80 REM TECLAS PARA MOVERSE: H,J
90 REM ************************
100 REM F PARA TERMINAR
110 IF A$="F" THEN GOTO 9999
120 PRINT AT F,C;" "
140 IF A$="H" THEN IF C>1 THEN C=C-1
150 IF A$="J" THEN C=C+1
180 PRINT AT F,C;"♟"
190 GOTO 60
```

Programa 2.



LA DEFINICION DE CARACTERES EN EL AMSTRAD

En el AMSTRAD, el nuevo carácter que hemos diseñado es asociado a un código ASCII y cuando queremos dibujar este carácter en pantalla, lo que deberemos hacer será poner:

PRINT CHR$(N)

siendo N el número del código ASCII al que hemos asignado el nuevo carácter que hemos definido o bien una variable numérica donde se encuentra guardado dicho número.

Hay que tener aquí en cuenta que al definir el nuevo carácter y asignarle un código ASCII, se pierde el carácter que tuviera asignado este código ASCII anteriormente. Por tanto, deberemos asignar el carácter al código ASCII de otro carácter que no vayamos a utilizar.

La instrucción para asignar la figura que nos hemos inventado a un código ASCII es la SYMBOL y se utiliza de la siguiente manera:

```
10 SYMBOL 250,24,24,90,90,126,60,60,126
```

Programa 3.

De los nueve números que se ponen a continuación de la palabra SYMBOL, el primero indica el código ASCII al que vamos a asignar el nuevo carácter y los ocho restantes son los números correspondientes a la lista de ceros y unos de cada una de las filas después de haberlas transformado de binario a decimal.

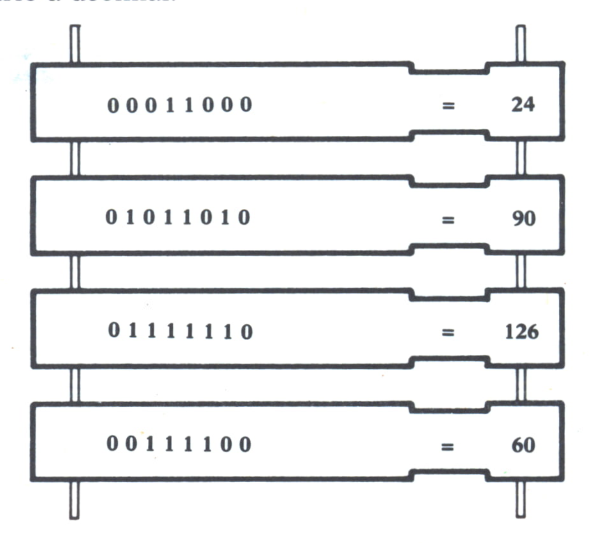


Fig. 7.

Después de ejecutar esta instrucción, en el código ASCII que hayamos puesto (250 en nuestro ejemplo), se habrá colocado el dibujo correspondiente a la nave espacial y cada vez que pongamos:

PRINT CHR$(250)

se dibujará en la pantalla una nave espacial.

Al igual que ocurría en el SPECTRUM, este código quedará con la figura asignada hasta que se le asigne otra con otra instrucción SYMBOL o se apague el ordenador.

Veamos cómo quedaría para el AMSTRAD el programa de mover la nave espacial con dos teclas después de haber ejecutado la instrucción SYMBOL reseñada anteriormente:

```
10 REM PROGRAMA PARA MOVER UN CARACTER CON 2 TECLAS
20 REM ********************************************
30 CLS
40 F=5:C=10
50 LOCATE C,F:PRINT CHR$(250)
60 A$=INKEY$
70 IF A$="" THEN GOTO 60
80 REM TECLAS PARA MOVERSE: H,J
90 REM ************************
100 REM F PARA TERMINAR
110 IF A$="F" THEN END
120 LOCATE C,F:PRINT " "
140 IF A$="H" THEN IF C>1 THEN C=C-1
150 IF A$="J" THEN C=C+1
180 LOCATE C,F:PRINT CHR$(250)
190 GOTO 60
```

Programa 4.

Cuando encendemos el ordenador AMSTRAD, tenemos la posibilidad de asignar los caracteres que nosotros diseñemos a los códigos ASCII comprendidos entre el 240 y el 255. Si queremos asignar caracteres a otro código, deberemos poner previamente la instrucción:

SYMBOL AFTER N

siendo N un número de código o una variable donde se encuentre guardado dicho número. A partir de que se ejecute esta instrucción, se podrán asignar los caracteres que hayamos definido nosotros, a los códigos ASCII comprendidos entre N y 255. De esta forma, ejecutando esta instrucción, podemos posteriormente asignar los caracteres a los códigos que queramos.

Si pusiésemos la instrucción:

SYMBOL AFTER 256

haríamos que no se pudieran asignar los caracteres a ningún código ASCII. Esto lo podríamos utilizar en algún caso en el que nos interesase que no se perdiera ninguno de los caracteres ASCII que vienen ya predefinidos en el ordenador al asignarles por algún error un carácter de los nuestros.

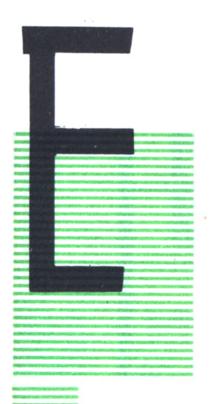
EJERCICIOS:

1. Diseñar una figura en una malla de 8 por 8 y realizar un programa para definirla en un carácter del ordenador. Hacer a continuación un programa, como el que se propuso en el capítulo anterior, para mover este carácter definido, a lo largo de la pantalla, hasta que se pulse una determinada tecla y se le pare.

LOS GRAFICOS PUNTO A PUNTO 7



INTRODUCCION



EN este último capítulo que vamos a dedicar a la realización de gráficos en el ordenador mediante un programa vamos a ver las técnicas que podemos llamar de dibujo propiamente dichas, ya que son las que se asemejan más a la forma manual de dibujar: realizar una figura a base de líneas continuas formadas por una serie de puntos consecutivos sin espacios entre ellos (con lo que conseguimos el efecto de línea continua).

LA PANTALLA PARA GRAFICOS

Cuando utilicemos las instrucciones para dibujar punto a punto que vamos a ver a continuación, la pantalla en la que vamos a hacer los dibujos se considera dividida en una malla al estilo de la que se representa en la siguiente figura:

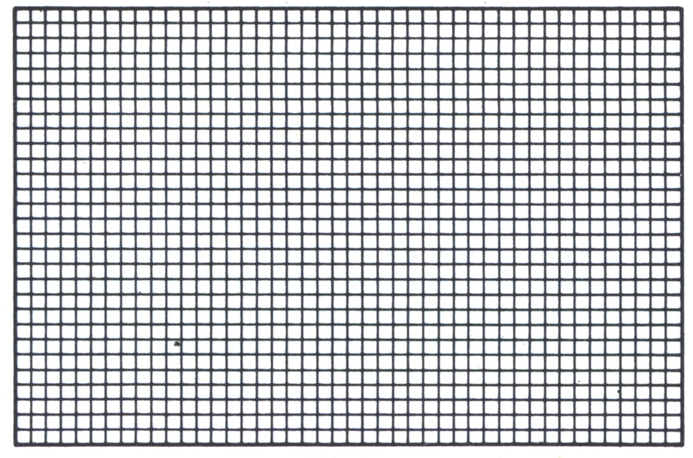


Fig. 1.

Cada uno de estos pequeños cuadritos es lo que vamos a llamar un punto. Cuando la división de la pantalla no sea muy grande estos puntos serán pequeños cuadraditos, mientras que cuando la división sea grande estos puntos se verán como tales. Hay ordenadores que permiten dividir la pantalla en dos o tres formas distintas (con puntos más grandes y puntos más pequeños), mientras que otros únicamente tienen una única división.

A continuación se adjunta un gráfico de las filas y columnas en las que se divide la pantalla para gráficos en cada uno de los ordenadores. En los que hay posibilidad de dividirla de varias formas así se indica:

	PUNTOS HORIZONTALES	PUNTOS VERTICALES
SPECTRUM	256	176
AMSTRAD	640	400 (Para los 3 modos)
COMMODORE	320	200
MS-DOS	320 640 640	200 — SCREEN 1 200 — SCREEN 2 400 — SCREEN 3

En el Spectrum y en el Amstrad se numeran de abajo a arriba los verticales y comienzan en cero

Fig. 2.

Estas divisiones, indudablemente, no se visualizan en pantalla. Habrá una serie de instrucciones, que vamos a estudiar en este capítulo, mediante las cuales podremos dibujar un punto en unas coordenadas determinadas que especifiquemos de la pantalla. Así, si en un programa se ejecuta la instrucción correspondiente a dibujar un punto en la columna 100, fila 150, la pantalla nos quedaría tal y como se muestra en la figura 3:

Aquellos ordenadores que tienen la posibilidad de dividir la pantalla de distintas maneras, también tendrán instrucciones correspondientes para realizar cada una de las divisiones.

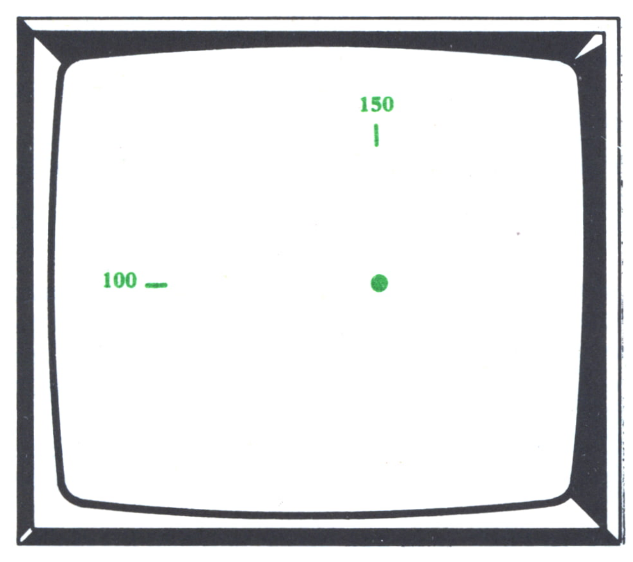


Fig. 3.



LAS INSTRUCCIONES PARA DIBUJAR PUNTOS

Para dibujar un punto en la pantalla, tal y como se ha dicho antes, hay dos instrucciones, dependiendo de los ordenadores en los que trabajemos:

a) *SPECTRUM, AMSTRAD, COMMODORE*

Utilizan la instrucción:

PLOT X,Y ,T

donde X indica la coordenada del punto a dibujar en el eje horizontal (columna), Y la coordenada en el eje vertical (fila) y T el color con el que queremos dibujar el punto correspondiente. Este valor T no se puede poner en el SPECTRUM. El punto se dibujará en este ordenador, con el color que hayamos especificado en la última instrucción INK. En el COMMODORE sólo se puede poner en T un 0 o un 1, según queramos que el punto nos lo dibuje con el color de la escritura o el mismo del fondo de la pantalla (es decir, nos lo borre). Cuando no se ponga nada en el lugar correspondiente de T, el punto se dibujará con el color especificado para la escritura en la última instrucción que se haya ejecutado para cambiar dicho color.

El Amstrad posee además la instrucción:

MOVE X,Y

que permite situar el cursor de gráficos en las coordenadas X e Y sin di-

bujar ningún punto en éstas. Funciona, como puede observarse, al igual que el LOCATE para la escritura de caracteres.

b) *MS-DOS (IBM)*

En estos ordenadores la instrucción para dibujar un punto en la pantalla es:

PSET (X,Y), color

X e Y tienen el mismo significado que en el PLOT visto anteriormente, y donde pone color se debe poner un número del 0 al 3 que corresponderá al color de ese número en la «paleta» (0 ó 1) que se haya especificado en la instrucción COLOR, para cuando se está en modo gráfico, que se vio en capítulos anteriores.

A diferencia de lo que ocurría con la instrucción PLOT, en ésta es obligatorio poner siempre el número del color con el que se quiere dibujar el punto.

Esta instrucción PSET es la que tienen también los equipos MSX.



LOS MODOS GRAFICOS

Una duda que posiblemente tenga el lector con lo que se ha visto hasta ahora es si estas instrucciones se pueden poner en cualquier parte del programa, entremezcladas con las instrucciones PRINT que hemos visto hasta ahora, o hay que usarlas aparte para dibujar.

Veamos lo que ocurre con esto en cada uno de los equipos:

a) *SPECTRUM*

El SPECTRUM tiene una división de la pantalla para escribir los caracteres y otra para dibujar los puntos tal y como se indica en el siguiente cuadro:

INSTRUCCIONES	FILAS	COLUMNAS
LOCATE, PRINT	22	32
PLOT	176	256

Fig. 4.

Cada vez que se ejecute en un programa una instrucción PLOT, se dibujará un punto en las coordenadas correspondientes y cada vez que se eje-

cute un PRINT, se escribirá lo que en él se indique en el lugar que corresponda.

Así, si pusiéramos el siguiente programa:

```
10 PLOT 150,20
20 PRINT AT 4,8;"HOLA"
30 PLOT 64,43
```

Programa 1.

la pantalla nos quedaría de la siguiente manera:

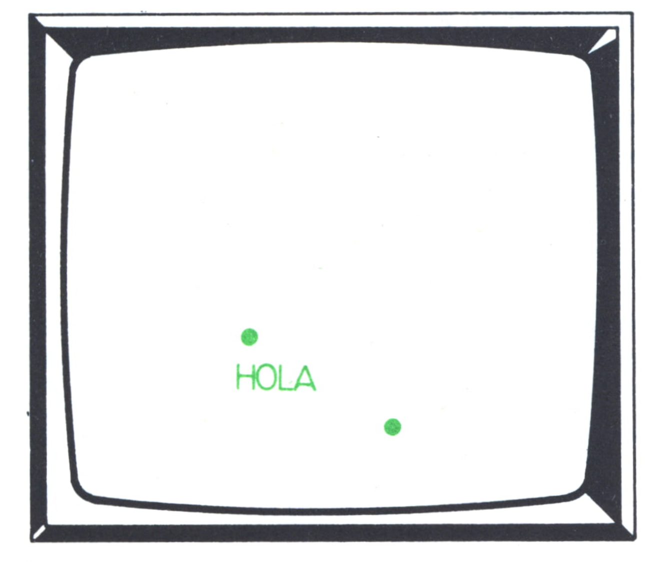


Fig. 5.

a) AMSTRAD

El AMSTRAD tiene lo que se llaman tres modos distintos: MODO 0, MODO 1 y MODO 2. En cada uno de estos tres modos hay unas coordenadas distintas para texto (instrucciones PRINT y LOCATE), siendo, sin embargo, las mismas para gráficos (instrucciones PLOT y MOVE) en los tres modos. Una vez que estamos en uno de los modos, el ordenador se comporta de la misma manera que lo hacía el SPECTRUM; es decir, por cada instrucción PLOT que pongamos se dibujará un punto en la pantalla de acuerdo con la división de la pantalla en el modo en que estemos, y cuando pongamos una instrucción PRINT se escribirá también de acuerdo con la división de la pantalla para texto.

Para cambiar de un modo a otro se utiliza la instrucción MODE. Esta instrucción tiene el siguiente formato:

MODE N

siendo N un número comprendido entre 0 y 2 o, como decimos siempre, una variable que contenga ese número (o una expresión cuyo resultado sea un número entre 0 y 2). Cuando un programa ejecuta una instrucción MODE cambia la pantalla al modo indicado en ésta, borrándose todo lo que hubiera escrito en el modo anterior y quedando preparada para recibir instrucciones PLOT, MOVE, PRINT o LOCATE con coordenadas correspondientes al nuevo modo.

A la hora de trabajar en el AMSTRAD con las instrucciones para hacer gráficos de puntos (PLOT y las que se estudien posteriormente dentro de su misma línea) hay que tener en cuenta las siguientes instrucciones:

— GRAPHICS PAPER: Funciona igual que PAPER, pero sirve para poner el color del fondo de la pantalla cuando se dibujen puntos mediante las instrucciones de gráficos.

— GRAPHICS PEN: Funciona igual que el PEN, pero para la escritura de puntos mediante las instrucciones de gráficos.

— CLG: Sirve para borrar la pantalla de gráficos.

c) *MS-DOS*

Los ordenadores con sistema operativo MS-DOS (IBM o compatibles) tiene tres posibles modos graficos cuyas coordenadas de pantalla se expresaron ya anteriormente.

Para poner la pantalla en cada uno de estos modos se debe poner la instrucción:

SCREEN N

siendo N un número comprendido entre 1 y 3. Al ejecutarse esta instrucción se pondrá la pantalla en el modo indicado, borrándose todo lo que hubiera escrito en ese momento. Estando en cualquiera de estos tres modos se pueden ejecutar tanto instrucciones de dibujar puntos (con las coordenadas correspondientes) como instrucciones para escribir caracteres (print); escribiéndose cada cosa que mandemos (puntos o caracteres) en el lugar correspondiente, de acuerdo con las coordenadas especificadas.

El número de filas y columnas para el texto (escritura de caracteres) es de 24 × 80 en todos los modos, excepto en el SCREEN 1, que es de 24 × 40.

Estos equipos MS-DOS poseen además el modo SCREEN 0, que es en el que se está cuando se enciende el ordenador y no se ejecuta ninguna instrucción SCREEN. Este modo es únicamente para texto y en él no se pueden utilizar las instrucciones de dibujo de puntos.

El siguiente programa, escrito para MS-DOS, haría lo que se indica en la figura:

```
10 SCREEN 1
20 PSET(100,150),1
30 PRINT "HOLA"
```

Programa 2.



Fig. 6.

Los ordenadores MSX tienen el modo SCREEN 1 del MS-DOS funcionando igual que éstos en este modo.

d) *COMMODORE*

El Commodore posee una única pantalla gráfica de 320 puntos en horizontal por 200 en vertical.

Es importante destacar aquí que en el SPECTRUM y en el AMSTRAD las coordenadas verticales se cuentan de abajo arriba, mientras que en el resto se hace al contrario.

Recordar que para las instrucciones de gráficos del Commodore es necesario que tenga el BASIC Simón (ampliación del BASIC del Commodore que también existe en el mercado).

Como resumen final del dibujo de puntos en los modos gráficos, se ofrecen dos programas que dibujan 30 puntos en la pantalla aleatoriamente de distintos colores. El primer programa es para equipos MS-DOS y el segun-

do para AMSTRAD. Se deja como ejercicio a los lectores que intenten realizarlo para el resto de los ordenadores, según las normas explicadas anteriormente.

```
10 REM PUNTOS ALEATORIOS MS-DOS
20 REM ************************
30 SCREEN 1
40 FOR I=1 TO 30
50 COL=INT(RND*3)+1
60 PSET(INT(RND*320)+1 ,INT(RND*200)+1),COL
70 NEXT I
```

Programa 3.

```
10 REM PUNTOS ALEATORIOS AMSTRAD
20 REM *************************
30 MODE 0
40 FOR I=1 TO 15
50 INK I,I
60 NEXT I
70 FOR I=1 TO 30
80 COL=INT(RND*15)+1
90 PEN COL
100 PLOT INT(RND*640),INT(RND*400)
110 NEXT I
```

Programa 4.



EL DIBUJO DE RECTAS

Hasta ahora hemos visto cómo se dibujan puntos en la pantalla, pero indudablemente los dibujos no están formados únicamente por puntos, sino que en la mayoría de las ocasiones queremos que nos haga figuras a base de rectas o circunferencias. Vamos a ver en este apartado cómo podemos conseguir realizar estas líneas.

La primera idea para dibujar una recta con las instrucciones que conocemos sería poner:

```
10 FOR I= 100 TO 200
20 PLOT I,100
30 NEXT I
```

Programa 5.

Estas instrucciones anteriores nos dibujarían 100 puntos desde el punto 100,100 al 200,100. Si ejecutamos este pequeño programa observaremos que esto se traduce en una línea horizontal desde el punto 100,100 al 200,100.

Utilizando esta misma técnica podríamos conseguir dibujar rectas en vertical o diagonal. Sin embargo, no es necesario poner estas instrucciones para dibujar rectas, ya que nuestro ordenador va a tener unas instrucciones específicas para esto, que vamos a ver a continuación:

El **SPECTRUM** dispone de la instrucción **DRAW** y el **AMSTRAD** de la **DRAWR** para trazar rectas. Ambas instrucciones funcionan igual y su formato es el siguiente:

DRAW
DRAWR } **incremento x, incremento y**

Fig. 7.

donde «incremento x» es el número que se sumará a la coordenada x del punto en el que se encuentra el cursor de gráficos actualmente e «incremento y» es el número que se sumará a la coordenada y. Tras sumar estos dos números a las coordenadas obtendremos las coordenadas de un nuevo punto. La instrucción trazará una recta desde el punto en el que se encuentra actualmente el cursor de gráficos hasta el nuevo punto que se ha obtenido como fruto de la suma antes indicada.

Así, si en el **SPECTRUM** ponemos:

```
10 PLOT 100,100
20 DRAW -25,50
```

Programa 6.

o en el **AMSTRAD**:

```
10 MOVE 100,100
20 DRAWR -25,50
```

Programa 7.

al ejecutarse estas instrucciones se nos dibujaría una recta desde el punto 100,100 al 75,150 tal y como se indica en la figura:

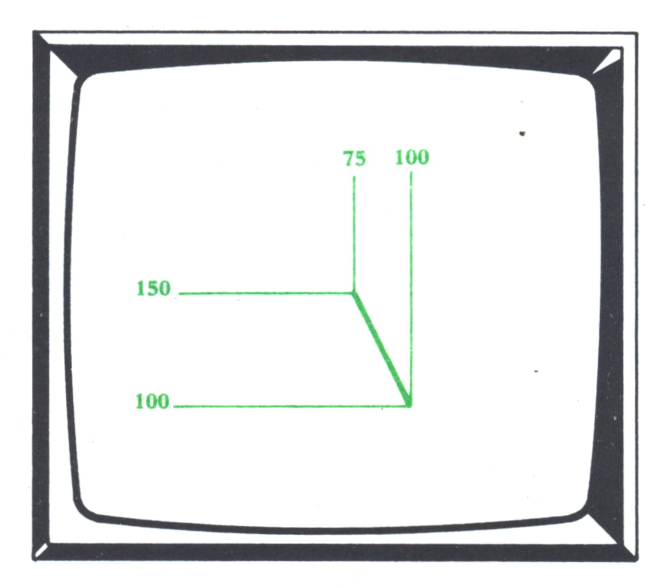


Fig. 8.

Es importante destacar aquí que la recta se dibuja desde el punto en el que se encuentra el cursor de gráficos. El cursor de gráficos se encontrará en el punto en el que se hiciera la última instrucción PLOT o MOVE. Cuando queramos trazar una recta a partir de un punto distinto deberemos poner previamente a la DRAW o la DRAWR una instrucción PLOT para situarnos en ese punto, como hemos hecho en el ejemplo expuesto anteriormente.

El AMSTRAD posee además una instrucción DRAW con el siguiente formato:

DRAW x,y

siendo x e y las coordenadas del punto en el que queremos que termine la recta que dibuje. Así, el programa para dibujar la recta anterior nos quedaría en el AMSTRAD utilizando la instrucción DRAW de la siguiente manera:

```
10 MOVE 100,100
20 DRAW   75,150
```

Programa 8.

El COMMODORE dispone de la instrucción LINE para el trazado de rectas, que tiene el siguiente formato:

LINE X1 , Y1 , X2 , Y2 , T

en donde X1 y Y1 son las coordenadas del punto en el que queremos que inicie la recta y X2 y Y2 las del punto en el que queremos que la termine. T tendrá el valor 1 cuando queramos la dibuje con el color definido para la escritura, mientras que tendrá el valor 0 cuando queramos que la dibuje con el mismo color del fondo de la pantalla (es decir, que la borre).

El MS-DOS tiene también esta misma instrucción LINE con el siguiente formato:

LINE (X1,Y1)-(X2,Y2),color

El funcionamiento es el mismo que para el COMMODORE. Si se omiten las coordenadas X1,Y1 se toma como punto de inicio de la recta aquel en el que se encuentre el cursor de gráficos en ese momento.

El MS-DOS tiene también la instrucción DRAW con el siguiente formato:

DRAW A$

donde A$ es una expresión alfanumérica que puede contener alguno de los cuatro valores que se expresan en el cuadro adjunto:

Un	**Dibuja una recta de n puntos hacia arriba**
Dn	**Dibuja una recta de n puntos hacia abajo**
Ln	**Dibuja una recta de n puntos hacia la izqda.**
Rn	**Dibuja una recta de n puntos hacia la dcha.**

n debe ser una constante numérica y las rectas se dibujan a partir de dónde se encuentre el cursor de gráficos

Fig. 9.

También se pueden poner dentro del valor alfanumérico varios valores de éstos separados por punto y coma, realizándonos en este caso las operaciones correspondientes a cada uno de ellos una detrás de otra.

Veamos un ejemplo para dibujar un cuadrado mediante esta instrucción:

```
10 SCREEN 1
20 PSET(150,50),1
30 DRAW"U40;R40;D40;L40"
```

Programa 9.

Con estas instrucciones vistas podemos realizar cualquier tipo de figuras que estén formadas a base de rectas. Como ejemplo final se pone un programa para dibujar la siguiente figura:

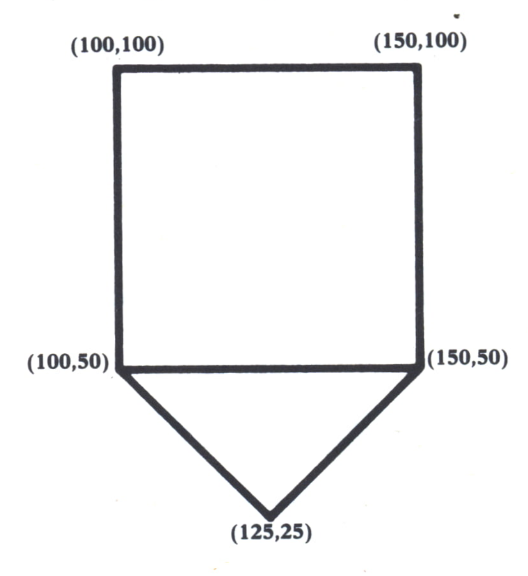


Fig. 10.

```
10 REM FIGURA PARA SPECTRUM
20 REM ********************
30 PLOT 100,100
40 DRAW 50,0
50 DRAW 0,-50
60 DRAW -50,0
70 DRAW 0,50
80 PLOT 100,50
90 DRAW 25,-25
100 DRAW  25,25
```

Programa 10.



EL DIBUJO DE CIRCUNFERENCIAS

Para dibujar circunferencias tenemos una instrucción común a todos los ordenadores, excepto el AMSTRAD: la instrucción CIRCLE tiene el formato:

CIRCLE X,Y,R

Cuando un programa ejecuta esta instrucción dibuja un círculo de centro las coordenadas X e Y y de radio el valor especificado en R. Para los equipos MS-DOS se debe poner esta instrucción con el formato:

CIRCLE (X,Y),R

En el AMSTRAD no existe esta instrucción y el dibujo de las circunferencias hay que realizarlo mediante el siguiente programa:

```
10 REM CIRCUNFERENCIAS PARA AMSTRAD
20 REM ****************************
30 CLS
40 INPUT "COORDENADAS DEL CENTRO"X,Y
50 INPUT "RADIO";R
60 CLS:CLG
70 MOVE X+R,Y
80 FOR A=0 TO 6.28 STEP 3.14/180
90 DRAW X+(R*COS(A)),Y+(R*SIN(A))
100 NEXT A
```

Programa 11.

Como final del capítulo se ofrece un programa que realiza un bonito dibujo en la pantalla a base de combinación de circunferencias. Está realizado para equipos MS-DOS, aunque con las pautas explicadas anteriormente no resultará difícil adaptarlo a cualquier otro equipo.

```
10 REM JUEGO CON CIRCUNFERENCIAS
20 REM *************************
30 SCREEN 1
40 FOR A= 100 TO 180  STEP 4
50 CIRCLE (A,A),50
60 NEXT A
```

Programa 12.

EJERCICIOS:

1. Realizar un programa para dibujar una figura geométrica a base de líneas mediante las pautas explicadas en el capítulo, y hacer que se desplace hacia la derecha al pulsar una tecla y hacia la izquierda al pulsar otra. (NOTA: Recordar que la operación de borrar la figura se puede hacer de manera sencilla dibujándola después de haber puesto como color de los gráficos el mismo del fondo de la pantalla.)

2. Realizar un trozo de programa que simule el movimiento de un proyectil a

lo largo de la pantalla. (NOTA: Mover un punto o dos puntos juntos de acuerdo con la técnica de movimiento relatada en el capítulo 5.)

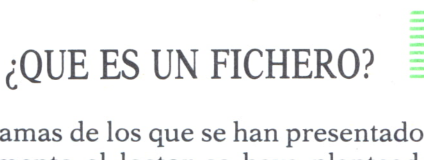
RESPUESTAS:

1.

```
10 REM MOVIMIENTO DIBUJO-SPECTRUM
20 REM **************************
30 CLS
40 P=10
45 GOSUB 1000
50 A$=INKEY$
60 IF A$="" THEN GOTO 50
65 INK 0 :GOSUB 1000
70 IF A$="Z" THEN P=P-10
80 IF A$="X" THEN P=P+10
85 INK 4
90 GOSUB 1000
100 GOTO 50
1000 REM SUBRUTINA PARA DIBUJAR UN CUADRADO A PARTIR DE
1010 REM LA COORDENADA HORIZONTAL P
1020 PLOT P,100
1030 DRAW 50,0
1040 DRAW 0,-50
1050 DRAW -50,0
1060 DRAW 0,50
1070 RETURN
```

Programa 13.

GESTION DE FICHEROS EN BASIC 8



¿QUE ES UN FICHERO?

EN numerosos programas de los que se han presentado hasta ahora, probablemente el lector se haya planteado un problema que no ha sabido resolver: cómo se pueden conservar los datos que el programa obtiene como fruto de las operaciones que realiza o los que le introducimos mediante distintas instrucciones INPUT. Toda la información que maneja un programa (los datos numéricos o alfanuméricos) está guardada en sus distintas variables, ya sean estas variables simples o listas o tablas, y cuando el programa termina o apagamos el ordenador ya no se puede volver a acceder a estos datos, ya que al acabar un programa se borran todos los datos que hubiera en sus variables.

La única forma que conocíamos hasta ahora de conservar de alguna manera los datos era listarlos por pantalla o por impresora, pero esto resulta evidentemente insuficiente, ya que si otro día quisiésemos ejecutar algún programa que utilizase los datos obtenidos en ese programa tendríamos que introducirle al nuevo programa todos esos datos mediante instrucciones INPUT, con la incomodidad que esto supone si es una gran cantidad de datos.

La idea de **FICHERO** nos va a solucionar este problema. Un **FICHERO** es un lugar de disco o cinta en el que se graban los datos que un programa tiene guardados en algunas de sus variables, mediante unas instrucciones específicas, y al que puede acceder otro programa o el mismo, mediante otras instrucciones, para leer los datos que se grabaron.

Así, si recordamos, por ejemplo, el programa que hicimos para controlar el stock de una serie de productos existentes en un almacén, cada vez que se comenzara a ejecutar el programa habría que introducir el stock existente de cada producto, ya que, por ejemplo, al finalizar de introducir los cambios existentes, obtener el listado con las nuevas existencias y acabar el programa y apagar el ordenador, la cantidad de stock de cada pro-

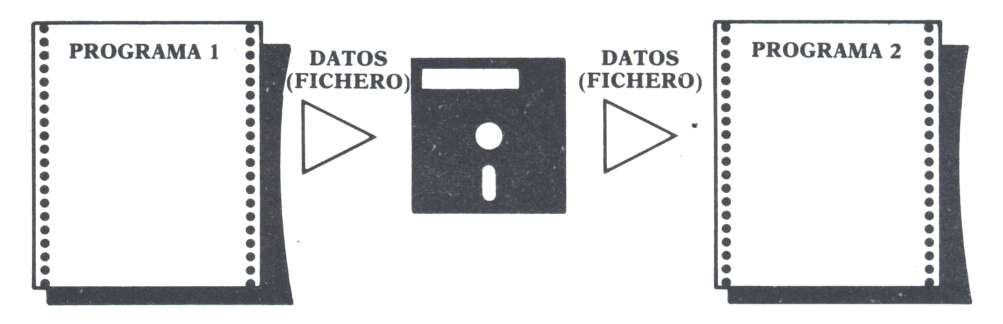


Fig. 1.

ducto que teníamos guardada en cada variable de la lista se nos borra. Si al día siguiente quisiésemos volver a ejecutar el programa, tendríamos que volver a meter en las variables de la lista las cantidades que teníamos del día anterior.

Utilizando ficheros, se nos solucionaría esto, ya que podríamos poner una instrucción para grabar esas cantidades en un fichero y otra para leerlas del fichero y pasarlas de nuevo a las variables de la lista.



TIPOS DE FICHEROS Y ESTRUCTURA

Indudablemente, los datos que grabamos en un fichero exigen que esto se haga de acuerdo a una determinada estructura que vamos a ver a continuación. Los ficheros que utilicemos pueden ser de dos tipos: secuenciales o de acceso directo.



LOS FICHEROS SECUENCIALES

Un fichero secuencial es aquel en el que los datos deben ser leídos en el mismo orden en que fueron grabados y comenzando siempre por el primero (en orden de secuencia).

Los ficheros se estructuran en lo que se llaman REGISTROS. Un registro de un fichero secuencial en BASIC estará formado por todos los datos que se graben de una sola vez (en una misma instrucción de las que veamos a continuación). Cada instrucción que ejecute un programa para grabar datos en un fichero secuencia, creará un nuevo registro en éste, inmediatamente después del último que se grabó, con los nuevos datos grabados.

REGISTRO	REGISTRO	REGISTRO
GARCIA 23	NAVARRO	15 12 19

Posible estructura de un fichero secuencial en el que se han grabado 3 registros con distintos datos cada uno

Fig. 2.

— Las instrucciones para manejar los ficheros secuenciales en BASIC son:

1) *Abrir el fichero*

En todos los programas en los que se vaya a utilizar un fichero, antes de realizar cualquier operación sobre él se debe ejecutar la instrucción OPEN (abrir), que sirve para indicarle al ordenador que vamos a usar ese fichero, especificando al mismo tiempo sus características.

Las características que se suelen especificar en un fichero al abrirle son las siguientes:

a) Tipo de fichero: Secuencial o de acceso directo.

b) Modo para el que se abre el fichero: Para leer datos de él o grabar datos en él.

c) Número del fichero: Todos los ficheros que se utilicen en un mismo programa deben llevar un número para identificarlos.

d) Nombre del fichero: Este será el nombre con el que el fichero se grabará en el disco o la cinta. Si hubiera varias unidades de disco o cinta habría que especificar aquí, además del nombre, la unidad donde se encuentra el disco o la cinta en la que queremos grabar.

Es importante destacar la diferencia existente entre el número de fichero y el nombre de éste. El nombre va a ser aquel con el que quede grabado en el disco y, por tanto, el que habrá que poner cada vez que queramos utilizar el fichero en cualquier programa. Aparte de esto, todos los ficheros que se utilicen en un mismo programa deben ser numerados con un número distinto para poderlos identificar luego, ya que en las instrucciones que se pongan para grabar o leer datos de ellos, se pone el número del fichero en el que se quiere hacer la operación. Así, un mismo fichero tendrá el MISMO nombre en cualquier programa que sea utilizado, pero puede tener DISTINTOS números, ya que si, por ejemplo, en un programa es abierto después de que ya hay cuatro más con los números del 1 al 4 se le asignará el número 5, mientras que si en otro programa sólo se le utiliza a él, se le puede abrir con el número 1 o con cualquier otro. El número de fichero en algunos manuales recibe también el nombre de canal.

Vistas las características que determinan un fichero, veamos los formatos de la instrucción **OPEN** en los distintos ordenadores:

— *COMMODORE 64:*

OPEN n. de fichero,1,modo,«nombre»

El 1 que se pone a continuación del número de fichero indica que el fichero es de acceso secuencial. Como en el **COMMODORE** la única opción posible es grabar en cinta de casette y ésta solo admite ficheros secuenciales, este 1 es fijo.

En modo se pondrá O (inicial de Output) si se abre el fichero para grabar datos en él e I (inicial de Input) si se abre para leer datos de él.

Por ejemplo, si en un programa aparece la instrucción:

20 OPEN 1,1,O,«NOMBRES»

significa que se abre un fichero con el número 1 para grabar datos en él.

— *MS-DOS (IBM O COMPATIBLES)*

OPEN «modo»,# n. de fichero,«nombre»

El significado de cada parámetro es el mismo que para el Commodore, teniendo en cuenta que cuando queramos abrir un fichero que **EXISTA** para añadirle nuevos datos lo debemos hacer poniéndole como modo «A». Si quisiésemos abrir el mismo fichero del ejemplo anterior deberíamos poner:

20 OPEN «O»,#1,«NOMBRES»

— *MSX*

OPEN «nombre» **FOR** modo **AS** n. de fichero

Para abrir el mismo fichero, se debería poner:

20 OPEN «NOMBRES» FOR O AS 1

— *AMSTRAD*

El **AMSTRAD** permite utilizar en un programa un solo fichero secuencial, en la cinta o en el disco, al que asocia el número 9. Este fichero se podrá abrir con las instrucciones:

OPENIN «nombre»

cuando lo queramos abrir para obtener los datos que tenga grabados.

OPENOUT «nombre»

cuando lo queramos abrir para grabar datos en él.

2) *Cerrar el fichero*

Cuando se hayan terminado de realizar todas las operaciones que se hagan en un programa con un fichero, se debe cerrar éste. Si no se hace esta operación, se pueden generar problemas de que no se queden correctamente grabados los datos.

La operación de cerrar un fichero se hace mediante la instrucción CLOSE que tiene el formato general:

CLOSE #n.fichero

En el AMSTRAD es en el único en el que esta instrucción varía. Se debe poner:

CLOSEIN

cuando se quiera cerrar el fichero que hemos abierto con OPENIN y:

CLOSEOUT

cuando se quiera cerrar el fichero que hayamos abierto con OPENOUT.

Notar que, mientras en todos los equipos podemos disponer en un mismo programa de todos los ficheros que queramos tener simultáneamente abiertos (bastará con que a cada uno le asignemos un número de fichero distinto) en AMSTRAD únicamente podemos tener un único fichero abierto, al que asigna el número de fichero 9. De todas formas, si en un programa AMSTRAD deseamos utilizar más de un fichero, bastará con que los usemos de uno en uno y antes de abrir otro, cerremos el anterior.

Así, si en un programa queremos utilizar los tres ficheros denominados: «TELEFONOS», «NOMBRES» y «AGENDA», en este orden, deberemos poner la secuencia de instrucciones:

OPENIN «TELEFONOS»

instrucciones correspondientes a lo que se quiera realizar con el fichero TELEFONOS

```
.................
CLOSEIN
OPENIN «NOMBRES»
..............
..............
CLOSEIN
OPENOUT «AGENDA»
.................
.................
CLOSEOUT
```

Recordar que en todas las instrucciones PRINT# e INPUT# que se utilicen se deberá poner como número de fichero el 9. Así, por ejemplo: PRINT#9, A$

3) *Grabar datos en el fichero*

Cuando abrimos un fichero para grabar datos en él (modo O), la instrucción que se emplea para ir grabando los datos es la PRINT#. El formato de esta instrucción PRINT# es:

PRINT #n. de fichero, lista de variables separadas por punto y coma

Cuando un programa ejecuta una instrucción PRINT#, crea un nuevo registro en el fichero cuyo número se especifica, formado por todos los datos contenidos en las variables que se ponen a continuación de PRINT#.

Veamos un primer programa de manejo de ficheros para guardar en un fichero los datos relativos a las señas de una serie de personas.

```
10 REM GRABACION DIRECCIONES FICHERO SECUENCIAL
20 REM ****************************************
30 CLS
40 OPEN "O",#1,"DIREC"
50 INPUT "TECLEA UN NOMBRE";N$
60 IF N$="" THEN GOTO 150
70 INPUT "TECLEA LOS APELLIDOS(SEPARADOS POR COMAS)";A1$,A2$
80 PRINT#1,N$;A1$;A2$
90 INPUT "TECLEA LA CALLE";C$
100 INPUT "NUMERO DE LA CALLE";N
110 PRINT #1,C$;N
120 INPUT "TECLEA EL TELEFONO";T
130 PRINT #1,T
140 GOTO 50
150 CLOSE #1
```

Programa 1.

El programa comentado anteriormente del control de stock de los artículos de un almacén quedaría de la siguiente manera si le añadimos la opción de grabar en un fichero secuencial las cantidades de cada artículo:

```
10 REM control de stock de un almacen
20 REM ******************************
30 DIM A(10)
40 CLS
50 PRINT "MENU" :PRINT
60 PRINT "INICIALIZAR STOCK DE CADA PRODUCTO            -->1"
70 PRINT "CAMBIAR STOCK DE UN PRODUCTO                  -->2"
80 PRINT "VISUALIZAR STOCK DE TODOS LOS PRODUCTOS       -->3"
85 PRINT "GRABAR STOCK DE CADA PRODUCTO EN UN FICHERO   -->4"
90 PRINT "TERMINAR                                      -->5"
100 PRINT :INPUT "TECLEA OPCION"; N
110 CLS
120 ON N GOSUB 1000,2000,3000, 4000
130 IF N<>5 THEN GOTO 50
```

```
140 END
1000 FOR I=1 TO 10
1010 PRINT "STOCK DEL PRODUCTO"; I
1020 INPUT A(I)
1030 NEXT I
1040 RETURN
2000 INPUT "N. DE PRODUCTO DEL QUE HA HABIDO CAMBIO";P
2010 INPUT "CAMBIO DE STOCK CON SU SIGNO";C
2020 A(P)=A(P)+C
2030 RETURN
3000 FOR I=1 TO 10
3010 PRINT "STOCK DEL PRODUCTO";I;"="; A(I)
3020 NEXT I
3030 RETURN
4000 OPEN "O",#1,"STOCK"
4010 FOR I=1 TO 10
4020 PRINT #1,A(I)
4030 NEXT I
4035 CLOSE#1
4040 RETURN
```

Programa 2.

En todos los programas que se pongan en este capítulo, la instrucción OPEN llevará el formato de los ordenadores MS-DOS. Para ejecutarlos en otros equipos, bastará con que se cambie por la correspondiente al equipo, de acuerdo con lo explicado anteriormente.

4) *Leer datos del fichero*

La instrucción más típica para leer los datos que tengamos grabados en un fichero es la INPUT#. Cuando nosotros abrimos un fichero que ya tengamos en el disco o la cinta, e indicamos que lo abrimos para leer datos de él, en la primera instrucción INPUT# que ejecute el programa referida a ese fichero, leerá el primer registro, en la segunda el segundo registro y así sucesivamente (recordar que cada registro está formado por todos los datos que se grabaron en una misma instrucción PRINT#). Estos datos que se leen con la instrucción INPUT# y que están almacenados en un mismo registro deben encontrarse en el fichero SEPARADOS POR COMAS, por lo que cuando en una instrucción PRINT# se graben varios datos que queramos luego posteriormente leer mediante la instrucción INPUT# debemos GRABAR ENTRE MEDIAS DE ELLOS UNA COMA. Por ejemplo:

PRINT#1,C$;«,»;B;«,»;A$

El formato de la instrucción INPUT# es el siguiente:

INPUT#n. de fichero, lista de variables separadas por comas

Las variables que pongamos después del número de fichero separadas por comas serán aquéllas en las que se guarden los datos que se lean. Es importante tener aquí en cuenta que las variables que pongamos para leer los datos deben ser del mismo número y del mismo tipo que las que pusimos cuando se grabaron los datos en el fichero, ya que si no nos dará error por no corresponderse los datos que hay y que hemos mandado leer con las variables que hemos puesto y donde hemos dicho que queremos que nos los guarde.

Veamos como ejemplo de esto, el programa del control de stock de los artículos del almacén, en el que hemos puesto una nueva posibilidad: la de leer el stock de cada artículo que teníamos guardado en el fichero.

```
10 REM control de stock de un almacen
20 REM ******************************
30 DIM A(10)
40 CLS
50 PRINT "MENU" :PRINT
60 PRINT "INICIALIZAR STOCK DE CADA PRODUCTO              -->1"
70 PRINT "CAMBIAR STOCK DE UN PRODUCTO                    -->2"
80 PRINT "VISUALIZAR STOCK DE TODOS LOS PRODUCTOS         -->3"
85 PRINT "GRABAR STOCK DE CADA PRODUCTO EN UN FICHERO     -->4"
87 PRINT "LEER DEL FICHERO EL STOCK DE CADA PRODUCTO      -->5"
90 PRINT "TERMINAR                                        -->6"
100 PRINT :INPUT "TECLEA OPCION"; N
110 CLS
120 ON N GOSUB 1000,2000,3000, 4000,5000
130 IF N<>6 THEN GOTO 50
140 END
1000 FOR I=1 TO 10
1010 PRINT "STOCK DEL PRODUCTO"; I
1020 INPUT A(I)
1030 NEXT I
1040 RETURN
2000 INPUT "N. DE PRODUCTO DEL QUE HA HABIDO CAMBIO";P
2010 INPUT "CAMBIO DE STOCK CON SU SIGNO";C
2020 A(P)=A(P)+C
2030 RETURN
3000 FOR I=1 TO 10
3010 PRINT "STOCK DEL PRODUCTO";I;"="; A(I)
3020 NEXT I
3030 RETURN
4000 OPEN "O",#1,"STOCK"
4010 FOR I=1 TO 10
4020 PRINT #1,A(I)
4030 NEXT I
4035 CLOSE#1
4040 RETURN
5000 OPEN "I",#2,"STOCK"
5020 FOR I=1 TO 10
```

```
5030 INPUT #2, A(I)
5040 NEXT I
5050 PRINT " DATOS LEIDOS Y ALMACENADOS EN LA LISTA A(10)"
5060 CLOSE #2
5070 RETURN
```

Programa 3.

Dos datos que hayan sido grabados sin comas de separación son considerados por la instrucción INPUT# como un solo dato formado por la concatenación de los dos.

— *Otras instrucciones para leer datos en un fichero*

*LINE INPUT #

Tiene el formato:

LINE INPUT#n. de fichero, var. alfanumérica

Esta instrucción guarda en la variable alfanumérica que pongamos todos los datos que haya grabados en el registro del fichero secuencial que toque leer, siempre que sean alfanuméricos, en el mismo orden en que estén, independientemente de que hayan sido grabados como uno o más datos. Así, si en un fichero hemos grabado los datos con la instrucción:

PRINT# A$;N$

y queremos leerlos más tarde mediante instrucciones INPUT#, deberemos poner en éstas dos variables alfanuméricas (además de haberlos grabado separados por comas). Con la instrucción LINE INPUT#, sin embargo, se pondría una variable alfanumérica, y, si por ejemplo, en el fichero que toca leer están grabados los datos de la figura:



Fig. 3.

en la variable se guardaría el valor: «GARCIAPEREZ».

*INPUT$ y GET#

Estas dos instrucciones nos permiten una manera distinta de leer los datos que tenemos grabados en un fichero secuencial: carácter a carácter en vez de como datos completos. Cuando en un fichero se utilizan para

leer las instrucciones INPUT$ o GET#, el programa se olvida de los registros y considera el fichero como una fila de caracteres uno detrás de otro. La instrucción INPUT$ tiene el siguiente formato:

var. alfanumérica=INPUT$(N,#n. de fichero)

En la variable alfanumérica que pongamos al principio, se guardarán los N caracteres que haya en el fichero después del último que se leyó.

Es importante destacar que cuando un programa graba un fichero, al final de cada registro, coloca un carácter RETURN para indicar el final del registro. Cuando leemos mediante instrucciones INPUT$ este carácter se lee y almacena como uno más entre el último del registro y el primero del siguiente, ya que con esta instrucción no se tienen en consideración los registros. De esta forma, en todos los programas que leamos datos con esta instrucción hay que tratar de controlar esto para que no nos almacene como dato este carácter que no lo es tal. El siguiente programa puede ser un ejemplo de cómo se podría evitar esto:

```
100 REM SE SUPONE QUE EL FICHERO FUE ABIERTO ANTERIORMENTE
190 B$=""
200 A$=INPUT$(1,#2)
210 IF A$=CHR$(13) THEN  GOTO 300
220 B$=B$+A$
230 GOTO 200
300 REM AQUI B$ CONTIENE UN REGISTRO COMPLETO
```

Programa 4.

El código ASCII 13 es el que se corresponde con el carácter RETURN, aunque siempre se aconseja consultar la tabla de códigos ASCII del manual de nuestro ordenador por si fuese otro número de código.

La instrucción GET# realiza la misma función que INPUT$. Si se empleara, la línea 200 debería escribirse:

200 GET#2,A$

Si el fichero contiene datos numéricos se debe utilizar INPUT# porque estas instrucciones al leer un carácter cada vez, harían muy difícil la reconstrucción de los números.

Como resumen final de los ficheros secuenciales, se ofrece un programa que lee los datos grabados de las direcciones de una serie de personas mediante el programa realizado anteriormente.

```
10 REM LECTURA DE UN CONJUNTO DE DIRECCIONES GRABADAS MEDIANTE
20 REM EL PROGRAMA ANTERIOR
30 REM ***************************************************
```

```
35 REM Habría que modificar del programa anterior las instrucciones
36 REM PRINT# y escribir una coma de separación entre cada dato que se escribe
40 CLS
50 INPUT "NUMERO DE DIRECCIONES QUE SE QUIEREN OBTENER";N
55 OPEN "I",#1,"DIREC"
60 FOR I= 1 TO N
70 INPUT#1,N$,A1$,A2$
80 PRINT "NOMBRE Y APELLIDOS:";N$,A1$,A2$
90 INPUT #1,C$,N
100 PRINT "DOMICILIO:";C$,N
110 INPUT #1,T
120 PRINT "TELEFONO";T
130 NEXT I
140 CLOSE #1
150 REM Existe la posibilidad de no necesitar pedir cuantos registros
160 REM se quieren leer sino leer todos hasta que se acaben.
170 REM En el último apartado del tema se estudia esta cuestión.
```

Programa 5.

Si cambiáramos la instrucción INPUT# por LINE INPUT# en aquella instrucción en la que se lee el nombre de la persona, en la variable que pusiéramos nos guardaría cosas como: «ANGEL,GARCIA,PEREZ»; por ejemplo.



LOS FICHEROS EN EL SPECTRUM

El **SPECTRUM** utiliza ficheros secuenciales, tal y como los hemos estudiado, solamente con las unidades Microdrive (pequeñas casetes que trabajan en bloques como los discos). En las unidades de casete únicamente se pueden grabar datos guardados en vectores o matrices mediante la instrucción **SAVE DATA**. Por ejemplo:

100 SAVE «NOMBRES» DATA N$( )

guardaría en la cinta los datos contenidos en la lista **N$** bajo el nombre de NOMBRES.

Para leer estos datos y guardarlos en otra nueva lista de otro programa que suponemos que se llama **A$** tendríamos que poner:

200 LOAD «NOMBRES» DATA A$( )



LOS FICHEROS DE ACCESO DIRECTO

Los ficheros de acceso directo son aquéllos de los que podemos leer los datos que hayamos grabado en el orden que queramos, independientemente del orden en que hayan sido grabados.

Estos ficheros sólo pueden ser utilizados en discos. Aquí vamos a ver las instrucciones para usar ficheros de acceso directo en los equipos **IBM** (sistema operativo **MS-DOS**), que utilizan discos de acceso directo.

La técnica para poder acceder a cualquier dato del fichero consiste en numerar los registros que se graben para luego acceder a ellos especificando el número del que queremos leer.

A diferencia de los ficheros secuenciales, en los ficheros de acceso directo TODOS LOS REGISTROS DEBEN TENER LA MISMA ESTRUCTURA. Es decir, si nosotros definimos que los registros van a estar formados por 3 datos alfanuméricos, en todos los registros que grabemos en el fichero, debemos introducir 3 datos alfanuméricos.

Este fichero de acceso directo tendría una estructura que podría ser de la forma:



Los datos que guardemos no tendrán siempre la longitud máxima. El resto se rellena con blancos

Fig. 4.

En los ficheros de acceso directo no se pueden grabar datos numéricos. Si quisiésemos hacerlo, tendríamos que grabarlos como caracteres (los datos numéricos se convierten en caracteres mediante la función STR$) y luego, después de leerlos, volverlos a pasar a números mediante la función VAL.

Vamos a ver a continuación los pasos que habría que dar para utilizar un fichero de acceso directo (en MS-DOS):

1) ABRIR el fichero mediante la instrucción OPEN vista para ficheros secuenciales poniendo como modo la letra «R».

2) DECLARAR la estructura que van a tener todos los registros del fichero. Esto se hace mediante la instrucción FIELD, que tiene el siguiente formato:

FIELD #n. de fichero, longitud1 AS variable1, longitud2 AS variable2,....

Se debe poner en esta instrucción una expresión: longitud AS variable por cada dato alfanumérico que queramos de que conste el registro.

En cada una de estas expresiones deberemos poner donde pone longitud, la longitud total del dato correspondiente y donde pone variable, el NOMBRE DE UNA VARIABLE ALFANUMERICA QUE NO SE HAYA UTILIZADO NI SE VAYA A UTILIZAR EN EL PROGRAMA PARA OTRA COSA y que ya veremos más adelante para qué sirve. Deberemos poner, por tanto, una variable por cada dato que queramos que tenga el registro del fichero.

Así, si un programa ejecuta la instrucción:

FIELD #3, 25 AS N$, 30 AS M$

después de haber ejecutado una instrucción de abrir un fichero de acceso directo con el número 3, significará que el registro va a constar de dos datos alfanuméricos y que al primero se le asocia la variable N$ y va a tener una longitud total de 25 caracteres y al segundo la variable M$ teniendo una longitud total de 30 caracteres.

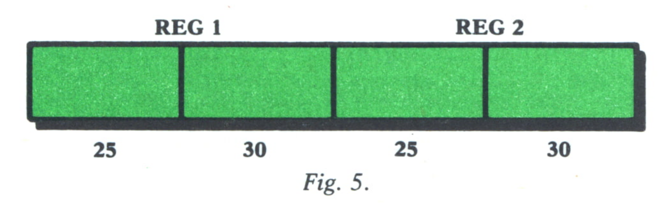


Fig. 5.

Cada vez que se abre un fichero y antes de empezar a grabar o leer datos por primera vez se debe poner esta instrucción FIELD. Si se abriese un fichero ya existente (para leer los datos grabados o grabar nuevos) se deberá poner en la instrucción FIELD el mismo formato de registro que tuviera las veces anteriores que fue utilizado.

3) Para GRABAR datos en el fichero:

Cuando queremos grabar los datos en un fichero de acceso directo, deberemos poner en primer lugar por cada dato del registro que queramos grabar en el fichero, la instruccion LSET. Esta instrucción tiene el mismo formato que la LET; debiéndose poner en el lugar de la variable, la variable asociada al dato que se puso en la instrucción FIELD.

Es importante notar que siempre hay que grabar y leer registros enteros.

Después de haber asignado a las variables asociadas los datos que queremos grabar en el registro del fichero que lo vamos a hacer, se debe poner la instrucción PUT#. Esta instrucción tiene el formato:

PUT#n. de fichero,n. de registro.

Cuando un programa ejecute esta instrucción, grabará en el registro cuyo número se especifique los datos contenidos en las variables asociadas a cada dato y cuyo valor les ha sido asignado mediante la instrucción LSET.

Así, si después de haber puesto la instrucción FIELD del ejemplo anterior, ponemos las instrucciones:

```
100 LSET N$="2322"
110 LSET M$="PEPE"
120 PUT#3,15
```

Programa 6.

en el registro número 15 del fichero se nos guardarían los datos especificados.

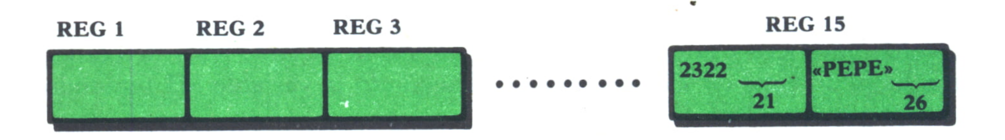


Fig. 6.

4) Para LEER datos grabados en un fichero de acceso directo:

Cuando queremos leer los datos que hay grabados en un determinado registro de un fichero de acceso directo, se pone la instrucción GET# cuyo formato es el siguiente:

GET#n. de fichero, n. de registro

Al ejecutarse esta instrucción se pasan a las variables asociadas a cada dato (las que especificamos en FIELD) los valores de los datos del registro del fichero cuyo número hemos indicado en la instrucción GET#. Luego, estos valores podemos ya obtenerlos de estas variables pudiéndolas manejar como si fueran variables normales.

Siguiendo con el ejemplo puesto anteriormente, si en ese mismo programa en el que habíamos abierto el fichero de acceso directo y habíamos declarado la estructura de su registro mediante la instrucción FIELD especificada, se ejecuta la instrucción:

GET#3,20

se pasarán a las variables N$ y M$ el primer y segundo dato guardados en el registro 20 del fichero respectivamente.

4) CERRAR el fichero

Los ficheros de acceso directo se cierran con la instrucción CLOSE de la misma forma que se cerraban los secuenciales:

CLOSE#n. de fichero

Como ejemplo final del empleo de ficheros de acceso directo, se adjunta a continuación el programa para reservar plazas en un tren de la manera más lógica como debe hacerse este programa: guardando el dato correspondiente a cada plaza (si está ocupada o no) en un fichero, de manera que cuando termine el programa y apaguemos el ordenador no se destruyan estos datos, y la reserva pueda durar más de un día, ya que al día siguiente, al encender el ordenador, continuamos teniendo en el fichero los datos correspondientes.

```
10 REM RESERVA DE PLAZAS EN 3 TRENES
20 REM ****************************
30 OPEN "R",#1,"TREN1"
40 OPEN "R",#2,"TREN2"
50 OPEN "R",#3,"TREN3"
60 FIELD#1, 1 AS A$
70 FIELD#2, 1 AS B$
80 FIELD#3, 1 AS C$
90 REM CADA TREN TENDRA 10 PLAZAS
100 CLS
110 INPUT "N. DE TREN EN EL QUE SE QUIERE RESERVAR";T
120 IF T=0 THEN  GOTO 220
130 FOR I=1 TO 10
140 GET#T,I
150 ON T GOTO 160,170,180
155 REM Una "0" en el registro indica que esa plaza está ocupada
160 IF A$="0" THEN GOTO 190 ELSE LSET A$="0":PUT#1,I:PRINT "RESERVADA PLAZA";I
     :GOTO 110
170 IF B$="0" THEN GOTO 190 ELSE LSET B$="0":PUT#2,I:PRINT "RESERVADA PLAZA";I
     :GOTO 110
180 IF C$="0" THEN GOTO 190 ELSE LSET C$="0":PUT#3,I:PRINT "RESERVADA PLAZA";I
     :GOTO 110
190 NEXT I
200 PRINT " NO HAY PLAZAS EN ESE TREN"
210 GOTO 110
220 CLOSE #1
230 CLOSE #2
240 CLOSE #3
```

Programa 7.



EL NUMERO DE REGISTROS DE UN FICHERO

El número de registros de un fichero, ya sea secuencial o de acceso directo, no está limitado y puede tener tantos como nosotros le pongamos.

Existe una función cuyo nombre es **EOF** y que tiene como argumento el número de un fichero secuencial. Esta función dará como resultado un 1 si en el fichero no quedan más registros por leer y un 0 si aún quedan registros. Es muy útil utilizarla cuando estemos leyendo los datos de un fichero secuencial, no sabemos cuántos registros hay y queremos leerlos todos hasta que se acaben. Seguidamente vemos el trozo de programa que nos haría esto.

```
10 REM LECTURA DE TODAS LAS DIRECCIONES GRABADAS MEDIANTE
20 REM EL PROGRAMA ANTERIORMENTE EXPUESTO
30 REM ***********************************************
35 REM Habría que modificar del programa anterior las instrucciones
36 REM PRINT# y escribir una coma de separación entre cada dato que se escribe
40 CLS
55 OPEN "I",#1,"DIREC"
70 INPUT#1,N$,A1$,A2$
```

```
80 PRINT "NOMBRE Y APELLIDOS:";N$,A1$,A2$
90 INPUT #1,C$,N
100 PRINT "DOMICILIO:";C$,N
110 INPUT #1,T
120 PRINT "TELEFONO";T
122 REM SI TODAVIA HAY MAS LEER OTRO
125 IF EOF(1)=0 THEN GOTO 70
140 CLOSE #1
```

Programa 8.

EJERCICIOS:

1. Hacer un programa para almacenar en un fichero los gastos que se han producido en una determinada empresa, pudiendo ser éstos relativos a diez conceptos distintos. Otro programa pedirá, cada vez que se ejecute, uno de los diez conceptos y la cantidad gastada de éste, acumulándola a la que ya hubiera. Hacer también otro programa para que a final de mes obtenga un listado del total de los gastos producidos en cada uno de los conceptos.

2. Hacer un programa que almacene en un fichero secuencial los nombres de una serie de personas junto con sus gastos. Realizar luego otro programa que, a partir del fichero creado por el programa anterior, liste por pantalla los nombres de cada una de las personas y pregunte si hay que añadirle algún gasto más. Si la respuesta es positiva, este programa deberá sumar este gasto al total que tenía y listar por impresora los gastos totales de cada persona (haya habido que aumentarle algo o no).

3. Hacer un programa parecido al anterior, pero suponiendo que las personas son alumnos de una clase y que lo que hay que guardar en el fichero es la nota de la evaluación para luego listarla mediante otro programa cuando se necesite.

4. Imaginar un programa que lleve la gestión de los clientes de un hotel.

Una vez conocidas las instrucciones fundamentales del lenguaje Basic, se plantea la cuestión de la realización de programas que resuelvan problemas o aplicaciones que se nos presentan diariamente en el trabajo, en casa o en los estudios.

Este libro trata de mostrar cómo se podrían realizar algunas de estas aplicaciones, estudiando diversas estructuras que proporciona el lenguaje Basic (como las subrutinas) y viendo las ideas fundamentales para realizar gráficos en pantalla mediante un programa y para almacenar datos en discos o cintas mediante los ficheros.